

PPPPPPPPPPPP		AAAAAAAAAA	TTTTTTTTTTTTTTTT	CCCCCCCCCCCC	HHH	HHH
PPPPPPPPPPPP		AAAAAAAAAA	TTTTTTTTTTTTTTTT	CCCCCCCCCCCC	HHH	HHH
PPPPPPPPPPPP		AAAAAAAAAA	TTTTTTTTTTTTTTTT	CCCCCCCCCCCC	HHH	HHH
PPP	PPP	AAA	TTT	CCC	HHH	HHH
PPP	PPP	AAA	TTT	CCC	HHH	HHH
PPP	PPP	AAA	TTT	CCC	HHH	HHH
PPP	PPP	AAA	TTT	CCC	HHH	HHH
PPP	PPP	AAA	TTT	CCC	HHH	HHH
PPP	PPP	AAA	TTT	CCC	HHH	HHH
PPPPPPPPPPPP		AAA	TTT	CCC	HHHHHHHHHHHHHHHH	HHH
PPPPPPPPPPPP		AAA	TTT	CCC	HHHHHHHHHHHHHHHH	HHH
PPPPPPPPPPPP		AAA	TTT	CCC	HHHHHHHHHHHHHHHH	HHH
PPP		AAAAAAAAAAAAAAAA	TTT	CCC	HHH	HHH
PPP		AAAAAAAAAAAAAAAA	TTT	CCC	HHH	HHH
PPP		AAAAAAAAAAAAAAAA	TTT	CCC	HHH	HHH
PPP		AAA	TTT	CCC	HHH	HHH
PPP		AAA	TTT	CCC	HHH	HHH
PPP		AAA	TTT	CCC	HHH	HHH
PPP		AAA	TTT	CCC	HHH	HHH
PPP		AAA	TTT	CCCCCCCCCCCC	HHH	HHH
PPP		AAA	TTT	CCCCCCCCCCCC	HHH	HHH
PPP		AAA	TTT	CCCCCCCCCCCC	HHH	HHH

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```
PPPPPPPP      AAAAAA      TTTTTTTTTT      EEEEEEEEEEE      NN      NN      CCCCCCCC
PPPPPPPP      AAAAAA      TTTTTTTTTT      EEEEEEEEEEE      NN      NN      CCCCCCCC
PP      PP      AA      AA      TT      EE      NN      NN      CC
PP      PP      AA      AA      TT      EE      NN      NN      CC
PP      PP      AA      AA      TT      EE      NNNN      NN      CC
PP      PP      AA      AA      TT      EE      NNNN      NN      CC
PPPPPPPP      AA      AA      TTT      EEEEEEEEE      NN      NN      CC
PPPPPPPP      AA      AA      TT      EEEEEEEEE      NN      NN      CC
PP      AAAAAAAAAA      TT      EE      NN      NNNN      CC
PP      AAAAAAAAAA      TT      EE      NN      NNNN      CC
PP      AA      AA      TT      EE      NN      NN      CC
PP      AA      AA      TT      EE      NN      NN      CC
PP      AA      AA      TT      EEEEEEEEEEE      NN      NN      CCCCCCCC
PP      AA      AA      TT      EEEEEEEEEEE      NN      NN      CCCCCCCC
```

```
LL      IIIIII      SSSSSSSS
LL      IIIIII      SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLLLL      IIIIII      SSSSSSSS
```

```
1 0001 0 MODULE PATENC (
2 L 0002 0 %IF %VARIANT EQL 1
3 0003 0 %THEN
4 0004 0 ADDRESSING_MODE (EXTERNAL = LONG_RELATIVE, NONEXTERNAL = LONG_RELATIVE),
5 0005 0 %FI
6 0006 0 IDENT = 'V04-000'
7 0007 0 ) =
8 0008 1 BEGIN
9 0009 1
10 0010 1 ++
11 0011 1 *****
12 0012 1 *
13 0013 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
14 0014 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
15 0015 1 * ALL RIGHTS RESERVED.
16 0016 1 *
17 0017 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
18 0018 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
19 0019 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
20 0020 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
21 0021 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
22 0022 1 * TRANSFERRED.
23 0023 1 *
24 0024 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
25 0025 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
26 0026 1 * CORPORATION.
27 0027 1 *
28 0028 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
29 0029 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
30 0030 1 *
31 0031 1 *
32 0032 1 *****
33 0033 1
34 0034 1 --
35 0035 1
36 0036 1 ++
37 0037 1 FACILITY: PATCH
38 0038 1 VAX INSTRUCTION ENCODER.
39 0039 1
40 0040 1 Version: V03-002
41 0041 1
42 0042 1 History:
43 0043 1 Author:
44 0044 1 Kevin Pammett, April 1977: Version 01
45 0045 1
46 0046 1 MODIFIED BY:
47 0047 1
48 0048 1 V03-003 MTR0011 Michael T. Rhodes 26-Jul-1982
49 0049 1 Modify immediate mode operand generation to allow I^#literal
50 0050 1 to be used with a byte context operand.
51 0051 1
52 0052 1 V03-002 MTR0006 Michael T. Rhodes 07-JUN-1982
53 0053 1 Add I^#literal functionality to routines GET NEXT_TOKEN,
54 0054 1 GET_OPERAND, and ENC_OPERAND. The context flag I_HAT_SEEN
55 0055 1 has been added to PATGEN.REQ and is reset in only two places.
56 0056 1 The end of encoding an absolute literal in ENC_OPERAND and in
57 0057 1 the end of command processing in PAT$SET_CONTEXT.
```

PATENC
V04-000

B 11
16-Sep-1984 00:40:15
14-Sep-1984 12:52:32

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[PATCH.SRC]PATENC.B32;1 (1)

Page 2

PAT
V04

58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

0058 1
0059 1
0060 1
0061 1
0062 1
0063 1
0064 1
0065 1
0066 1
0067 1
0068 1
0069 1
0070 1
0071 1
0072 1

V03-001 MTR0004 Michael T. Rhodes 24-MAR-1982
Fix endless loop conditions in routine GET_OPERAND,
within loops located at OPERATOR_CODE and at GET_PATHNAME,
which were caused by not advancing the instruction string
pointer. Fixes SPR #11-43576.

V02-022 KDM0052 Kathleen D. Morse 28-APR-1981
Fix bug that found any address ending in 2D illegal.

V02-021 PCG0001 Peter George 02-FEB-1981
Add require statement for LIB\$:PATDEF.REQ

FFF

FFF

```
: 74      0073 1 FORWARD ROUTINE
: 75      0074 1      PAT$INS_ENCODE,
: 76      0075 1      GET_NEXT_TOKEN,
: 77      0076 1      GET_LABEL : NOVALUE,
: 78      0077 1      GET_OPCODE,
: 79      0078 1      GET_OPERAND,
: 80      0079 1      ENC_OPERAND,
: 81      0080 1      ASSUME_AT_PC,
: 82      0081 1
: 83      0082 1      INST_OUTPUT,
: 84      0083 1      OPCODE_MATCH,
: 85      0084 1      MAR_GET_LEX,
: 86      0085 1      ADD_LEX_T_OPRND : NOVALUE,
: 87      0086 1      PAT$REDUCE_INS : NOVALUE,
: 88      0087 1      PAT$RESOLVE_INS : NOVALUE;
: 89      0088 1
: 90      0089 1 LIBRARY 'SYSSLIBRARY:LIB.L32';
: 91      0090 1 REQUIRE 'SRC$:SYSLIT.REQ';
: 92      0140 1 REQUIRE 'SRC$:VXSMAC.REQ';
: 93      0205 1 REQUIRE 'SRC$:PATGEN.REQ';
: 94      0427 1 REQUIRE 'SRC$:PATPCT.REQ';
: 95      0467 1 REQUIRE 'SRC$:PATTER.REQ';
: 96      0674 1 REQUIRE 'SRC$:PATOK.REQ';
: 97      0744 1 REQUIRE 'SRC$:VAXOPS.REQ';
: 98      0958 1 REQUIRE 'LIB$:PATDEF.REQ';
: 99      1012 1 REQUIRE 'LIB$:PATMSG.REQ';
: 100     1186 1 REQUIRE 'SRC$:SYSSER.REQ';
```

```
: Encode an instruction
: Extract next token from input stream.
: Extracts a label from instruction string i
: Extract opcode from instruction string.
: Acquires a lexical operand
: Encode an operand reference.
: Temporary 'look ahead' routine
:   to implement <operand> ::= <number>
: Output bytes of instruction.
: Find opcode in OPINFO table.
: Gets a MARS lexeme
: Annexes a lexical string onto an operand s
: Reduces expressions and symbols in instruc
: Resolves forward references for command
```

```
: Literals and macros related to opcodes
! Defines literals
! PATCH error reporting
! Macros for diagnostic output.
```

PATENC
V04-000

D 11
16-Sep-1984 00:40:15
15-Sep-1984 22:50:49

VAX-11 Bliss-32 V4.0-742
_ \$255\$DUA28:[PATCH.SRC]SYSSER.REQ;1

Page 4
(1)

PAT
V04

: R1218 1
: R1219 1
: R1220 1
: R1221 1
: R1222 1

SWITCHES LIST (SOURCE);

EXTERNAL ROUTINE
PAT\$fa0_out;

! formats a line and outputs to the terminal

```
101 1268 1 REQUIRE 'SRC$:PREFIX.REQ';
102 1456 1 REQUIRE 'SRC$:PATPRE.REQ';
103 1619 1 REQUIRE 'SRC$:BSTRUC.REQ';
104 1695 1 REQUIRE 'SRC$:DLLNAM.REQ';
105 1753 1
106 1754 1 EXTERNAL ROUTINE
107 1755 1     PAT$BUILD_PATH,
108 1756 1     PAT$DELETE_PATH : NOVALUE,
109 1757 1     PAT$DEFINE_SYM,
110 1758 1     PAT$FILL_BUF : NOVALUE,
111 1759 1     PAT$FIND_SYM,
112 1760 1     PAT$FIND_VAL,
113 1761 1     PAT$FREE_RELEASE : NOVALUE,
114 1762 1     PAT$FREEZ,
115 1763 1     PAT$SUBST_INS,
116 1764 1     RAD50,
117 1765 1     SYS$FAOL : ADDRESSING_MODE(Absolute),
118 1766 1     PAT$FAO_PUT : NOVALUE,
119 1767 1     PAT$MAR_GET_LEX,
120 1768 1     PAT$OUT_PUT : NOVALUE,
121 1769 1     PAT$REG_MATCH;
122 1770 1
123 1771 1
124 1772 1 EXTERNAL
125 1773 1     PAT$GL_SYMTBPTR,
126 1774 1     PAT$GL_OLDLABLS,
127 1775 1     PAT$GL_TEMP_BUF : BLOCK[,BYTE],
128 1776 1     PAT$GL_BR_DISPL,
129 1777 1     PAT$GL_CONTEXT : BITVECTOR,
130 1778 1     PAT$GL_ERRCODE,
131 1779 1     PAT$CP_OUT_STR : REF VECTOR[,BYTE],
132 1780 1     PAT$GB_MOD_PTR : REF VECTOR[,BYTE],
133 1781 1     PAT$GB_OPINFO : OPCODE_TBL,
134 1782 1     PAT$GB_OPINFO1 : BLOCKVECTOR[SIZOPINFO1,OPTSIZE,BYTE],
135 1783 1     PAT$GB_OPINFO2 : BLOCKVECTOR[SIZOPINFO2,OPTSIZE,BYTE],
136 1784 1     PAT$GB_ALIAS : BLOCKVECTOR[SIZALIAS,ALTSIZE,BYTE],
137 1785 1     PAT$GL_BUF_SIZ,
138 1786 1     PAT$GL_FWRCHD;
139 1787 1
140 1788 1 OWN
141 1789 1     BYTES_IN_OPRND;
142 1790 1
143 1791 1 LITERAL
144 1792 1     PAT$K_BR_RANGE = 2,
145 1793 1     MAX_REG = 15,
146 1794 1     MAX_BUF_SIZ = 512,
147 1795 1
148 1796 1 !++
149 1797 1 ! Return values for routine GET_OPERAND.
150 1798 1 !--
151 1799 1     OPR_FOUND = 1,
152 1800 1     NO_MORE_OPR = 0,
153 1801 1     OPR_ERROR = 2,
154 1802 1     OPR_FORW_REF = 3,
155 1803 1
156 1804 1 !++
157 1805 1 ! Bit masks for operand length indicators: B^,W^,L^, and [.
```

! Define structure macro
! Defines ASD structure

! Builds a path name structure
! Deletes a path name structure
! Defines a user symbol
! Routine to allocate and fill a buffer
! Find symbol in a doubly-linked symbol table
! Find value for symbol in a doubly-linked symbol table
! Deallocates free storage
! Allocates and zeros some free storage
! Routine to create branch instruction substitute
! Routine to convert ASCII to RAD50
! System service to do formatted output
! Formatted ASCII output
! Lexical scanner
! Actually write out lines to tty
! See if a char string is a reference to a register.

! Pointer to current symbol table listhead entry
! Pointer to old label table listhead entry
! Temporary instruction buffer
! Branch displacement that did not fit
! Context bits for PATCH commands
! Error code
! Points into output string

! General OPcode information structure
! Single byte OPcodes
! Double byte OPcodes
! OPcode alias table
! Data vector that describes opcodes
! Forward reference table listhead

! Number of encoded bytes in current operand

! Error code for branch displacement out of range
! Maximum number of registers - 1
! Maximum buffer size for pathname and operand

! Operand found and successfully reduced
! No more operands
! Expression reduction error found in operand
! Forward reference inside operand

```

: 158
: 159
: 160
: 161
: 162
: 163
: 164
: 165

1806 1 : Note that the masks for B^, W^ and L^ are also the number of bytes
1807 1 : to be included in the operand by these indicators. [ always indicates
1808 1 : context indexed and always donates one additional byte.
1809 1 :
1810 1 B_HAT_MASK = 1,
1811 1 W_HAT_MASK = 2,
1812 1 L_HAT_MASK = 4,
1813 1 CNT_IDX_MASK = 8;

: B^ mask
: W^ mask
: L^ mask
: Context indexed mask

```



```
167 1814 1 GLOBAL ROUTINE PAT$INS_ENCODE( INST_CS, OUT_BYTE_STREAM, OUT_PC, ASM_DIR_TBL, BUFFER_DSC ) =
168 1815 1
169 1816 1 **
170 1817 1
171 1818 1 FUNCTIONAL DESCRIPTION:
172 1819 1
173 1820 1 This routine examines an ascii stream which it is passed a pointer to,
174 1821 1 and tries to come up with the instruction byte stream this would
175 1822 1 correspond to.
176 1823 1
177 1824 1 CALLING SEQUENCE:
178 1825 1
179 1826 1 PAT$INS_ENCODE ();
180 1827 1
181 1828 1 INPUTS:
182 1829 1
183 1830 1 INST_CS - A pointer to the counted string which
184 1831 1 the user input as the supposed instruction.
185 1832 1
186 1833 1 OUT_BYTE_STREAM - The address of where to put the
187 1834 1 counted instruction stream generated.
188 1835 1 OUT_PC - The PC (memory address) where the instruction
189 1836 1 will eventually reside. This is used only for
190 1837 1 PC-relative address calculations.
191 1838 1 ASM_DIR_TBL - Address of descriptor for assembler directive table.
192 1839 1 BUFFER_DSC - String descriptor for current encoded instruction buffer.
193 1840 1
194 1841 1 IMPLICIT INPUTS:
195 1842 1
196 1843 1 PAT$GB_OPINFO - Data vector which contains the instruction
197 1844 1 mnemonics and related information.
198 1845 1
199 1846 1 OUTPUTS:
200 1847 1
201 1848 1 The counted instruction (byte) stream is put in the byte vector
202 1849 1 pointed to by the input parameter.
203 1850 1
204 1851 1 IMPLICIT OUTPUTS:
205 1852 1
206 1853 1 NONE.
207 1854 1
208 1855 1 ROUTINE VALUE:
209 1856 1
210 1857 1 TRUE or FALSE, depending on whether the encoding
211 1858 1 'worked' or not.
212 1859 1
213 1860 1 SIDE EFFECTS:
214 1861 1
215 1862 1 None.
216 1863 1
217 1864 1 --
218 1865 1
219 1866 2 BEGIN
220 1867 2
221 1868 2 MAP
222 1869 2 OUT_BYTE_STREAM : REF VECTOR[,BYTE],
223 1870 2 INST_CS : REF VECTOR[,BYTE],
```

```
224 1871 2 ASM DIR_TBL : REF BLOCK[,BYTE],
225 1872 2 BUFFER_DSC : REF BLOCK[,BYTE];
226 1873 2
227 1874 2 LOCAL
228 1875 2 COUNT,
229 1876 2 OPR_FLAG,
230 1877 2 ASD_TBL_ENTRY : BLOCK[ASD$C_SIZE, BYTE],
231 1878 2 TOKEN,
232 1879 2 ACT_NUM_OPRNDS,
233 1880 2 PC_START,
234 1881 2 INSTR_STRING : VECTOR[NO_OF_INP_CHARS,BYTE],
235 1882 2
236 1883 2
237 1884 2 INST_STG_DESC : BLOCK [8, BYTE],
238 1885 2 OPINFO_PTR : REF BLOCK[OPSIZE, BYTE],
239 1886 2 OUT_BYTE_PTR : REF VECTOR[,BYTE],
240 1887 2
241 1888 2
242 1889 2 BRANCH_SIZE,
243 1890 2 OPRNDS,
244 1891 2 OPRND_STG_DESC : BLOCK[12,BYTE],
245 1892 2 OPRND_BUF : VECTOR[MAX_BUF_SIZ,BYTE],
246 1893 2 OPCODE_NAME : REF VECTOR[,BYTE];
247 1894 2
248 1895 2 LITERAL
249 1896 2 JMP_OPCODE = %X'17';
250 1897 2 JSB_OPCODE = %X'16';
251 1898 2
252 1899 2 ++
253 1900 2 Remember starting PC for instruction in case substitution is required.
254 1901 2 --
255 1902 2 PC_START = .OUT_PC;
256 1903 2
257 1904 2 ++
258 1905 2 Copy the instruction string into a local area so that it won't be overwritten.
259 1906 2 Then put a terminator at the end of it as this will make recognizing the EOS
260 1907 2 easier for the scanner. Also initialize the pointer to the return buffer.
261 1908 2 --
262 1909 2 CH$MOVE( .INST_CS[0] + 1, INST_CS[0], INSTR_STRING );
263 1910 2 INSTR_STRING [INSTR_STRING [0] + 1] = 0;
264 1911 2 INSTR_STRING [0] = .INSTR_STRING [0] + 1;
265 1912 2 INST_STG_DESC [DSC$W_LENGTH] = .INSTR_STRING [0];
266 1913 2 INST_STG_DESC [DSC$A_POINTER] = INSTR_STRING [1];
267 1914 2 OUT_BYTE_PTR = OUT_BYTE_STREAM[1];
268 1915 2
269 1916 2 ++
270 1917 2 Check if there is a label on the instruction. If so, GET_LABEL removes it
271 1918 2 and enters it into the definition table.
272 1919 2 --
273 1920 2 GET_LABEL(INST_STG_DESC, .OUT_PC, TRUE);
274 1921 2
275 1922 2 ++
276 1923 2 Extract the opcode mnemonic from the instruction string and encode the
277 1924 2 corresponding opcode into the output byte stream. Pass the address of the
278 1925 2 local OPINFO_PTR, so that GET_OPCODE can initialize this. This is done in
279 1926 2 this way because it is GET_OPCODE who looks up the opcode (and hence finds
280 1927 2 the OPINFO record), but it is here that the other OPINFO information is
```

```
281 1928 2 needed. GET_OPCODE does not return if the opcode is unrecognized.
282 1929 2
283 1930 2 OPCODE_NAME = GET_OPCODE( INST_STG_DESC, OUT_BYTE_PTR, OPINFO_PTR, OUT_PC);
284 1931 2
285 1932 2
286 1933 2
287 1934 2
288 1935 2
289 1936 2
290 1937 2
291 1938 2
292 1939 2
293 1940 2
294 1941 2
295 1942 2
296 1943 2
297 1944 2
298 1945 2
299 1946 2
300 1947 2
301 1948 2
302 1949 2
303 1950 2
304 1951 2
305 1952 2
306 1953 2
307 1954 2
308 1955 2
309 1956 2
310 1957 2
311 1958 2
312 1959 2
313 1960 2
314 1961 2
315 1962 2
316 1963 2
317 1964 2
318 1965 2
319 1966 2
320 1967 2
321 1968 2
322 1969 2
323 1970 2
324 1971 2
325 1972 2
326 1973 2
327 1974 2
328 1975 2
329 1976 2
330 1977 2
331 1978 2
332 1979 2
333 1980 2
334 1981 2
335 1982 2
336 1983 2
337 1984 2

--
needed. GET_OPCODE does not return if the opcode is unrecognized.
--
OPCODE_NAME = GET_OPCODE( INST_STG_DESC, OUT_BYTE_PTR, OPINFO_PTR, OUT_PC);
--
++
GET_OPCODE has stuffed the opcode into the instruction stream (for all MACRO
instructions but not assembler directives). The pointer has been updated so
that further code will go into successive bytes.
--
IF (.OPINFO_PTR[OP_NUMOPS] NEQ ASM_DIR_OP)
THEN
    BEGIN
        ++
        The instruction-stream counted-string pointer now points to the
        beginning of the operand reference string, if there is one. This must
        be the case unless this opcode has no operands.
        --
        OPRNDS = .OPINFO_PTR [ OP_NUMOPS ];
        END
    ELSE
        ++
        Set a large number of operands as it is unknown how many operands
        an assembler directive may have. Also set up an entry in the
        assembler directive table. This is for PATCH verification output to
        the journal file.
        --
        BEGIN
            OPRNDS = MAXOPRND - 1;
            ASD_TBL_ENTRY[ASD$PC] = .OUT_PC;
            ASD_TBL_ENTRY[ASD$OPINFO] = .OPINFO_PTR;
            ASD_TBL_ENTRY[ASD$B_NUM_OPRND] = 0;
            END;
        ++
        Loop trying to extract each operand reference.
        --
        ACT_NUM_OPRNDS = 1;
        WHILE .ACT_NUM_OPRNDS LEQ .OPRND
        DO
            BEGIN
                ++
                Decide what type of branching the following routine will be allowed
                to use, if any. This is done now as the OPINFO information is
                available. For the same reason, the PC-relative context information
                is passed. Start by assuming the usual case.
                All assembler directive operands are handled as a special case.
                The number of bytes to be encoded is the OP_BR_TYPE value.
                --
                BRANCH_SIZE = NO_BR;
                IF (.ACT_NUM_OPRNDS EQL .OPRND) OR (.OPINFO_PTR[OP_NUMOPS] EQL ASM_DIR_OP)
                THEN
                    ++
                    Branching can only be considered for the last operand of an
                    instruction which has less than the maximum number of
                    operands. This assumption was made in constructing the
                    data structure (OPINFO) from which encoding (and decoding)
```

works. See PATINS.B32 for more information.

IF (.ACT_NUM_OPRNDS LSS MAXOPRND) OR (.OPINFO_PTR[OP_NUMOPS] EQL ASM_DIR_OP)
THEN

BEGIN
BRANCH_SIZE = .OPINFO_PTR[OP_BR_TYPE];
IF (.BRANCH_SIZE EQLU BR_LG)
THEN
BRANCH_SIZE = A_LONGWORD;
END;

++
Preprocess the operand, evaluating all expressions and symbols.

OPRND_STG_DESC[DSC\$W_LENGTH] = 0;
OPRND_STG_DESC[DSC\$A_POINTER] = OPRND_BUF;
OPRND_STG_DESC[DSC\$W_MAXLEN] = MAX_BUF_SIZE;
IF NOT (OPR_FLAG = GET_OPERAND(INST_STG_DESC, OPRND_STG_DESC, TRUE,
1 * .OPINFO_PTR[OP_CONTEXT(.ACT_NUM_OPRNDS)]))

THEN

BEGIN
IF (.OPR_FLAG EQLU OPR_ERROR)
THEN
RETURN(FALSE); ! Invalid expression
IF (.OPINFO_PTR[OP_NUMOPS] EQL ASM_DIR_OP)
THEN
EXITLOOP ! An assembler directive has indeterminate #
ELSE
BEGIN
SIGNAL(PAT\$INSOPRND+MSG\$K_INFO);
RETURN(FALSE); ! Insufficient operands
END;
END;

++
Check for forward referenced symbol. If this is the case,
update the pointers past this operand and create an entry
in the Forward Reference table for this operand.

IF (.OPR_FLAG EQLU OPR_FORW_REF)
THEN

BEGIN
LOCAL
POINTER : REF BLOCK[BYTE];
POINTER = PAT\$FREEZ((FWR\$C_SIZE + A_LONGWORD - 1)/A_LONGWORD);
POINTER[FWR\$FLINK] = .PAT\$GL_FWLRLD;
PAT\$GL_FWLRLD = POINTER;
POINTER[FWR\$PC] = .OUT_PC;
POINTER[FWR\$B_BUFOFF] = .BUFFER_DSC[DSC\$W_LENGTH] +
.OUT_BYTE_PTR - .OUT_BYTE_STREAM[1];
POINTER[FWR\$W_OPRNDLNG] = .OPRND_STG_DESC[DSC\$W_LENGTH];
POINTER[FWR\$A_OPRNDADR] = INST_CS[1] +
(.OPRND_STG_DESC[DSC\$A_POINTER] - INSTR_STRING[1]);
POINTER[FWR\$A_OPINFO] = .OPINFO_PTR;
COUNT = 0;
IF ((COUNT = .BYTES_IN_OPRND AND CONT_IND_X_MASK) NEQ 0)
THEN

```
395 2042 5 BEGIN
396 2043 5 COUNT = 1;
397 2044 5 BYTES_IN_OPRND = .BYTES_IN_OPRND - CONT_INDX_MASK;
398 2045 4 END;
399 2046 5 IF ((.OUT_BYTE_STREAM[1] EQL JMP_OPCODE) OR
400 2047 5 (.OUT_BYTE_STREAM[1] EQL JSB_OPCODE))
401 2048 4 THEN
402 2049 4 BYTES_IN_OPRND = L_HAT_MASK;
403 2050 5 IF (.BRANCH_SIZE NEQ NO_BR)
404 2051 4 THEN
405 2052 4 COUNT = .BRANCH_SIZE
406 2053 4 ELSE
407 2054 4 COUNT = .COUNT + .BYTES_IN_OPRND + 1;
408 2055 4 PCINTER[FWR$B_NUMBYTES] = .COUNT;
409 2056 4 PCINTER[FWR$B_NTHOPRND] = .ACT_NUM_OPRNDS;
410 2057 4 OUT_PC = .OUT_PC + .COUNT;
411 2058 4 OUT_BYTE_PTR = .OUT_BYTE_PTR + .COUNT;
412 2059 4 OPRND_STG_DESC[DSC$W_LENGTH] = 0;
413 2060 4 END
414 2061 3 ELSE
415 2062 3 !++
416 2063 3 ! Extract and encode one operand reference. Give up if this fails.
417 2064 3 !--
418 2065 4 BEGIN
419 2066 5 IF NOT (PAT$GL_ERRCODE = ENC_OPERAND( OPRND_STG_DESC, OUT_BYTE_PTR,
420 2067 5 1 ^ .OPINFO_PTR[OP_CONTEXT(.ACT_NUM_OPRNDS)],
421 2068 5 .BRANCH_SIZE, OUT_PC, .OPINFO_PTR))
422 2069 4 THEN
423 2070 5 BEGIN
424 2071 5 !++
425 2072 5 ! "Operand Syntax" error. If instruction substitution is enabled,
426 2073 5 ! then attempt to substitute other instructions for this branch
427 2074 5 ! instruction.
428 2075 5 !--
429 2076 5 IF (.PAT$GL_ERRCODE EQL PAT$K_BR_RANGE) AND
430 2077 5 (.PAT$GL_CONTEXT[INST_SUBST]) AND
431 2078 6 (.OPINFO_PTR[OP_NUMOPS] NEQ ASM_DIR_OP) ! Don't substitute for assembler dir
432 2079 5 THEN
433 2080 6 BEGIN
434 2081 6 OUT_BYTE_STREAM[0] = .OUT_BYTE_PTR - OUT_BYTE_STREAM[1];
435 2082 6 PAT$SUBST_INS(.OUT_BYTE_STREAM, .PC_START);
436 2083 5 END;
437 2084 5 RETURN(FALSE);
438 2085 4 END;
439 2086 3 END;
440 2087 3 !++
441 2088 3 ! Check that the operand was completely used. If not, error.
442 2089 3 !--
443 2090 3 IF (.OPRND_STG_DESC[DSC$W_LENGTH] NEQ 0)
444 2091 4 THEN
445 2092 3 RETURN(FALSE);
446 2093 3
447 2094 3 ACT_NUM_OPRNDS = .ACT_NUM_OPRNDS + 1;
448 2095 3 END;
449 2096 2
450 2097 2 !++
451 2098 2
```

```

452 2099 2 ! Check that there are not any extra operands.
453 2100 2 !--
454 2101 2 OPRND_STG_DESC[DSC$W_LENGTH] = 0;
455 2102 2 OPRND_STG_DESC[DSC$A_POINTER] = OPRND_BUF;
456 2103 2 OPRND_STG_DESC[DSC$W_MAXLEN] = MAX_BUF_SIZE;
457 2104 2 IF GET_OPERAND( INST_STG_DESC, OPRND_STG_DESC, TRUE, 0)
458 2105 2 THEN
459 2106 2 BEGIN
460 2107 2 SIGNAL(PAT$NUMOPRND$MSG$K_INFO, 2, .OPCODE_NAME, .OPRND$); ! Too many operands
461 2108 2 RETURN(FALSE);
462 2109 2 END;
463 2110 2
464 2111 2 !++
465 2112 2 ! Calculate the number of bytes in the 'instruction' stream and copy this value
466 2113 2 ! into the 0th byte of the output vector. This makes it now a counted byte stream.
467 2114 2 !--
468 2115 2 OUT_BYTE_STREAM[0] = .OUT_BYTE_PTR - OUT_BYTE_STREAM[1];
469 2116 2
470 2117 2 !++
471 2118 2 ! Now if this was an assembler directive. Compute the number of operands it
472 2119 2 ! had and enter that into the assembler directive entry. Then add this entry
473 2120 2 ! to the assembler directive table.
474 2121 2 !--
475 2122 2 ACT_NUM_OPRND$ = .ACT_NUM_OPRND$ - 1;
476 2123 2 IF (.OPINFO_PTR[OP_NUMOPS] EQL ASM_DIR_OP)
477 2124 2 THEN
478 2125 2 BEGIN
479 2126 2 ASD_TBL_ENTRY[ASD$B_NUM_OPRND] = .ACT_NUM_OPRND$;
480 2127 2 PAT$FILL_BUF(.ASM_DIR_TBL, ASD_TBL_ENTRY, ASD$C_SIZE);
481 2128 2 END;
482 2129 2
483 2130 2 !++
484 2131 2 ! This is the only place where this routine returns the OK completion status.
485 2132 2 ! If some error was encountered, then a message was signaled and an
486 2133 2 ! UNWIND or RETURN(FALSE) occurred. The only exception is branch of
487 2134 2 ! out of range. In this case instruction substitution has been attempted.
488 2135 2 !--
489 2136 2 RETURN(TRUE);
490 2137 1 END;
```

```

.TITLE PATENC
.IDENT \V04-000\

.PSECT _PAT$OWN,NOEXE,2

00000 BYTES_IN_OPRND:
.BLK 4

ISE$C_SIZE== 20
TXT$C_SIZE== 4
PAL$C_SIZE== 16
ASD$C_SIZE== 9
FWR$C_SIZE== 24
.EXTRN PAT$FAO_OUT, PAT$BUILD_PATH
.EXTRN PAT$DELETE_PATH
.EXTRN PAT$DEFINE_SYM, PAT$FILL_BUF
```

PAT
V04

				0188	31	00098	4\$:	BRW	19\$		
				58	D4	0009B	5\$:	CLRL	BRANCH_SIZE		1977
			59	53	D1	0009D		CMPL	ACT_NUM_OPRNDS, OPRNDS		1978
FFFFFFFE	8F	66	04	0B	13	000A0		BEQL	6\$		
				00	EC	000A2		CMPL	#0, #4, (R6), #2		
			06	1E	12	000AB		BNEQ	8\$		
				53	D1	000AD	6\$:	CMPL	ACT_NUM_OPRNDS, #6		1987
				0B	19	000B0		BLSS	7\$		
FFFFFFFE	8F	66	04	00	EC	000B2		CMPL	#0, #4, (R6), #2		
				0E	12	000BB		BNEQ	8\$		
	58	07	A5	04	EF	000BD	7\$:	EXTZ	#4, #4, 7(R5), BRANCH_SIZE		1990
			03	58	D1	000C3		CMPL	BRANCH_SIZE, #3		1991
				03	12	000C6		BNEQ	8\$		
			58	04	D0	000C8		MOVL	#4, BRANCH_SIZE		1993
				CD	B4	000CB	8\$:	CLRW	OPRND_STG_DESC		1999
		FF60	CD	AE	9E	000CF		MOVAB	OPRND_BUF, OPRND_STG_DESC+4		2000
		FF64	CD	8F	B0	000D5		MOVW	#512, OPRND_STG_DESC+8		2001
			53	02	78	000DC		ASHL	#2, ACT_NUM_OPRNDS, R1		2003
	50	51	04	51	EF	000E0		EXTZV	R1, #4, (R6), R0		
		66	01	50	78	000E5		ASHL	R0, #1, R4		
		54		54	DD	000E9		PUSHL	R4		
				01	DD	000EB		PUSHL	#1		2002
				CD	9F	000ED		PUSHAB	OPRND_STG_DESC		
				CD	9F	000F1		PUSHAB	INST_STG_DESC		
		00000000V	EF	04	FB	000F5		CALLS	#4, GET_OPERAND		
			5A	50	D0	000FC		MOVL	R0, OPR_FLAG		
			20	5A	E8	000FF		BLBS	OPR_FLAG, 10\$		
			02	5A	D1	00102		CMPL	OPR_FLAG, #2		2006
				18	13	00105		BEQL	9\$		
FFFFFFFE	8F	66	04	00	EC	00107		CMPL	#0, #4, (R6), #2		2009
				86	13	00110		BEQL	4\$		
				8F	DD	00112		PUSHL	#7176835		2014
		00000000G	00	01	FB	00118		CALLS	#1, LIB\$SIGNAL		
				0167	31	0011F	9\$:	BRW	23\$		2015
			03	5A	D1	00122	10\$:	CMPL	OPR_FLAG, #3		2024
				03	13	00125		BEQL	11\$		
				009D	31	00127		BRW	17\$		
				06	DD	0012A	11\$:	PUSHL	#6		2029
		00000000G	EF	01	FB	0012C		CALLS	#1, PAT\$FREEZ		
			60	EF	D0	00133		MOVL	PAT\$GL_FWRLHD, (POINTER)		2030
		00000000G	EF	50	D0	0013A		MOVL	POINTER, PAT\$GL_FWRLHD		2031
		04	A0	AC	D0	00141		MOVL	OUT_PC, 4(POINTER)		2032
			51	BC	3C	00146		MOVZWL	@BUFFER_DSC, R1		2034
			51	AE	C0	0014A		ADDL2	OUT_BYTE_PTR, R1		
14	A0		51	57	C3	0014E		SUBL3	R7, R1, 20(POINTER)		
		08	A0	CD	B0	00153		MOVW	OPRND_STG_DESC, 8(POINTER)		2035
			51	CD	9E	00159		MOVAB	INSTR_STRING+1, R1		2037
	51	FF60	CD	51	C3	0015E		SUBL3	R1, OPRND_STG_DESC+4, R1		
			51	AC	C0	00164		ADDL2	INST_CS, R1		
		0C	A0	A1	9E	00168		MOVAB	1(R1), 12(POINTER)		2036
		10	A0	55	D0	0016D		MOVL	R5, 16(POINTER)		2038
				52	D4	00171		CLRL	COUNT		2039
		52	00000000'	EF	8F	CB	00173	BICL3	#-9, BYTES_IN_OPRND, COUNT		2040
				0A	13	0017F		BEQL	12\$		
			52	01	D0	00181		MOVL	#1, COUNT		2043
		00000000'	EF	08	C2	00184		SUBL2	#8, BYTES_IN_OPRND		2044
			17	67	91	0018B	12\$:	CMPB	(R7), #23		2046

			16		05	13	0018E	BEQL	13\$		
					67	91	00190	CMPB	(R7), #22	2047	
					07	12	00193	BNEQ	14\$		
		00000000'	EF		04	D0	00195	13\$: MOVL	#4, BYTES_IN_OPRND	2049	
					58	D5	0019C	14\$: TSTL	BRANCH_SIZE	2050	
			52		05	13	0019E	BEQL	15\$		
					58	D0	001A0	MOVL	BRANCH_SIZE, COUNT	2052	
					0C	11	001A3	BRB	16\$		
	51		52	00000000'	EF	C1	001A5	15\$: ADDL3	BYTES_IN_OPRND, COUNT, R1	2054	
			52	01	A1	9E	001AD	MOVAB	1(R1), COUNT		
		0A	A0		52	90	001B1	16\$: MOVB	COUNT, 10(POINTER)	2055	
		0B	A0		53	90	001B5	MOVB	ACT_NUM_OPRNDS, 11(POINTER)	2056	
		0C	AC		52	C0	001B9	ADDL2	COUNT, OUT_PC	2057	
		08	AE		52	C0	001BD	ADDL2	COUNT, OUT_BYTE_PTR	2058	
				FF5C	CD	B4	001C1	CLRW	OPRND_STG_DESC	2059	
					51	11	001C5	BRB	18\$	2024	
					55	DD	001C7	17\$: PUSHL	R5	2068	
				0C	AC	9F	001C9	PUSHAB	OUT_PC	2066	
				0110	8F	BB	001CC	PUSHR	#MZR4, R8>	2067	
				18	AE	9F	001D0	PUSHAB	OUT_BYTE_PTR	2066	
				FF5C	CD	9F	001D3	PUSHAB	OPRND_STG_DESC		
		00000000V	EF		06	FB	001D7	CALLS	#6, ENC_OPERAND		
		00000000G	EF		50	D0	001DE	MOVL	R0, PAT\$GL_ERRCODE		
			30		50	E8	001E5	BLBS	R0, 18\$		
			02	00000000G	EF	D1	001E8	CMPL	PAT\$GL_ERRCODE, #2	2076	
					6C	12	001EF	BNEQ	20\$		
FFFFFFFE	8F	64	00000000G	EF	04	E1	001F1	BBC	#4, PAT\$GL_CONTEXT+2, 20\$	2077	
		66		04	00	EC	001F9	CMPL	#0, #4, (R6), #-2	2078	
					59	13	00202	BEQL	20\$		
	08	BC	08	AE	57	83	00204	SUBB3	R7, OUT_BYTE_PTR, @OUT_BYTE_STREAM	2081	
					5B	DD	0020A	PUSHL	PC_START	2082	
				08	AC	DD	0020C	PUSHL	OUT_BYTE_STREAM		
		00000000G	EF		02	FB	0020F	CALLS	#2, -PAT\$SUBST_INS		
					71	11	00216	BRB	23\$	2084	
				FF5C	CD	B5	00218	18\$: TSTW	OPRND_STG_DESC	2091	
					6B	12	0021C	BNEQ	23\$		
					53	D6	0021E	INCL	ACT_NUM_OPRNDS	2095	
				FE70	31	00220	BRW	3\$		1966	
				FF5C	CD	B4	00223	19\$: CLRW	OPRND_STG_DESC	2101	
		FF60	CD	0C	AE	9E	00227	MOVAB	OPRND_BUF, OPRND_STG_DESC+4	2102	
		FF64	CD	0200	8F	B0	0022D	MOVW	#512, -OPRND_STG_DESC+8	2103	
			7E		01	7D	00234	MOVQ	#1, -(SP)	2104	
				FF5C	CD	9F	00237	PUSHAB	OPRND_STG_DESC		
				FF68	CD	9F	0023B	PUSHAB	INST_STG_DESC		
		00000000V	EF		04	FB	0023F	CALLS	#4, GET_OPERAND		
			16		50	E9	00246	BLBC	R0, 21\$		
					59	DD	00249	PUSHL	OPRND	2107	
				04	AE	DD	0024B	PUSHL	OPCODE_NAME		
					02	DD	0024E	PUSHL	#2		
		00000000G	00	006D8233	8F	DD	00250	PUSHL	#7176755		
					04	FB	00256	CALLS	#4, LIB\$SIGNAL		
					2A	11	0025D	BRB	23\$	2108	
	08	BC	08	AE	57	83	0025F	20\$: SUBB3	R7, OUT_BYTE_PTR, @OUT_BYTE_STREAM	2115	
					53	D7	00265	DECL	ACT_NUM_OPRNDS	2122	
FFFFFFFE	8F	66		04	00	EC	00267	CMPL	#0, #4, (R6), #-2	2123	
					13	12	00270	BNEQ	22\$		
		FC	AD		53	90	00272	MOVW	ACT_NUM_OPRNDS, ASD_TBL_ENTRY+8	2126	

PATENC
V04-000

C 12
16-Sep-1984 00:40:15
14-Sep-1984 12:52:32

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[PATCH.SRC]PATENC.B32;1
Page 16
(3)

PAT
V04

			09	DD	00276		PUSHL	#9		: 2127
		F4	AD	9F	00278		PUSHAB	ASD_TBL_ENTRY		:
		10	AC	DD	0027B		PUSHL	ASM_DIR_TBL		:
00000000G	EF		03	FB	0027E		CALLS	#3, PAT\$FILL_BUF		:
	50		01	D0	00285	22\$:	MOVL	#1, R0		: 2136
				04	00288		RET			:
			50	D4	00289	23\$:	CLRL	R0		: 2137
				04	0028B		RET			:

; Routine Size: 652 bytes, Routine Base: _PAT\$CODE + 0000

; R

```
492 2138 1 ROUTINE GET_NEXT_TOKEN ( INST_STG_DESC, LEX_BUFFER ) =
493 2139 1
494 2140 1 ++
495 2141 1 Functional Description:
496 2142 1
497 2143 1 Scan the input stream and extract the next token from it. This routine
498 2144 1 is similar to the one that the parser uses - it is just that this one
499 2145 1 does a little more, and is a little more selective about what it accepts.
500 2146 1
501 2147 1 Calling Sequence:
502 2148 1
503 2149 1 GET_NEXT_TOKEN (INST_STG_DESC, LEX_BUFFER)
504 2150 1
505 2151 1 Inputs:
506 2152 1
507 2153 1 INST_CS_PTR -Contains the address of the current instruction
508 2154 1 counted-string pointer. We pick up and update
509 2155 1 this pointer via this address.
510 2156 1 LEX_BUFFER -A pointer to where we can pass back a value
511 2157 1 which is associated with whatever token we
512 2158 1 discover. If this value is 0, we don't
513 2159 1 try to pass back anything.
514 2160 1
515 2161 1 Implicit Inputs:
516 2162 1
517 2163 1 PAT$GL_MOD_PTR - is used by the radix convert routine
518 2164 1 to convert numeric input.
519 2165 1
520 2166 1 Outputs:
521 2167 1
522 2168 1 None.
523 2169 1
524 2170 1 Implicit Outputs:
525 2171 1
526 2172 1 Via LEX_BUFFER, we pass back:
527 2173 1 -a counted byte stream in LEX_BUFFER if the token
528 2174 1 is LONG_VAL_TOKEN, WORD_VAL_TOKEN, ABS_LIT_TOKEN,
529 2175 1 or BYTE_VAL_TOKEN. Even though only <count> bytes
530 2176 1 of this value are valid as far as operand size is
531 2177 1 concerned, the value is written out as a (sign-extended)
532 2178 1 longword so that we can do arithmetic on it and worry
533 2179 1 about the size only when we want to extract it again.
534 2180 1 -a REGISTER number in a 1-byte field of LEX_BUFFER
535 2181 1 for REGISTER_TOKEN, INDEXING_TOKEN, or AT_REG_TOKEN.
536 2182 1 -A 5-byte sequence for BRANCH_TOKEN. The first byte
537 2183 1 of this is 1 => PC-relative type, or 0 => absolute
538 2184 1 type. The remaining 4 bytes are the
539 2185 1 longword representation of the branch operand.
540 2186 1
541 2187 1 Return Value:
542 2188 1
543 2189 1 The literal that stands for the token type we have
544 2190 1 extracted. This is either one of those mentioned
545 2191 1 above, or it is BAD_TOKEN.
546 2192 1 --
547 2193 1
548 2194 2 BEGIN
```

```
549 2195 2
550 2196 2 MAP
551 2197 2 INST_STG_DESC : REF BLOCK [ , BYTE],
552 2198 2 LEX_BUFFER : REF VECTOR[ , BYTE];
553 2199 2
554 2200 2 LOCAL
555 2201 2 LEX_STG_DESC : BLOCK [12, BYTE], ! Lexeme string descriptor
556 2202 2 TOKEN_BUFFER : VECTOR[ CHS_PER_LEXEME+1, BYTE ],
557 2203 2 GARBAGE : VECTOR[ CHS_PER_LEXEME, BYTE ],
558 2204 2 INST_CS : REF VECTOR[ , BYTE],
559 2205 2 REG_NUM,
560 2206 2 MODE,
561 2207 2 RETURN_PTR,
562 2208 2 TOKEN_TYPE : BYTE; ! The returned value.
563 2209 2
564 2210 2 MACRO
565 M 2211 2 PASS_BACK( BYTES, VALUE ) = ! Used to pass back the indicated
566 2212 2 (.RETURN_PTR)<0,BYTES*BITS_PER_BYTE> = VALUE %; ! number of bytes, if requested.
567 2213 2
568 2214 2 !++
569 2215 2 ! If a value is associated with the token extracted, this value is returned via
570 2216 2 ! the pointer, LEX_BUFFER. If, however, this pointer is 0, then the caller does
571 2217 2 ! not want such information. To avoid having to check this several times, then,
572 2218 2 ! a local pointer, RETURN_PTR, is used to point either to where the user wants
573 2219 2 ! the value, or to some local garbage location.
574 2220 2 !--
575 2221 2 IF ((RETURN_PTR = .LEX_BUFFER) EQL 0)
576 2222 2 THEN
577 2223 2 RETURN_PTR = GARBAGE;
578 2224 2
579 2225 2 !++
580 2226 2 ! Pick up the current instruction counted-string pointer.
581 2227 2 !--
582 2228 2 LEX_STG_DESC [DSC$W_LENGTH] = 0;
583 2229 2 LEX_STG_DESC [DSC$A_POINTER] = TOKEN_BUFFER;
584 2230 2 LEX_STG_DESC [DSC$W_MAXLEN] = CHS_PER_LEXEME + 1;
585 2231 2
586 2232 2 !++
587 2233 2 ! Extract the next token and take the appropriate action.
588 2234 2 !--
589 2235 2 TOKEN_TYPE = MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC);
590 2236 2
591 2237 2 !++
592 2238 2 ! Sometimes a keyword token is returned when the user
593 2239 2 ! really input a symbolic name of some sort.
594 2240 2 !--
595 2241 2 IF (.TOKEN_TYPE GTR 0) AND (.TOKEN_TYPE LEQ KEYWORD_RANGE)
596 2242 2 THEN
597 2243 2 TOKEN_TYPE = ALPHA_STR_TOKEN;
598 2244 2 TOKEN_TYPE = ? SELECTONE .TOKEN_TYPE OF
599 2245 2
600 2246 2 SET
601 2247 2
602 2248 2 [HASH_TOKEN]: ! '#'
603 2249 2 !++
604 2250 2 ! Extract the literal and pass it back.
605 2251 2 !--
```

```

606 2252 4      IF (MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC) NEQ DIGIT_STR_TOKEN)
607 2253 3      THEN
608 2254 3          BAD_TOKEN
609 2255 3      ELSE
610 2256 4          BEGIN
611 2257 4              PASS_BACK (LONG_LENGTH, .(.LEX_STG_DESC [DSC$A_POINTER]));
612 2258 4              ABS_CIT_TOKEN
613 2259 3          END;
614 2260 3
615 2261 3
616 2262 3      [AT SIGN_TOKEN, : 'a', For @#const, @(reg), and @displ(reg)
617 2263 3      PLUS_TOKEN,   : '+', for (reg)+
618 2264 3      MINUS_TOKEN,  : '-', For -(sp) Etc.
619 2265 3      COMMA_TOKEN,   : ',', For arg1, arg2, Etc.
620 2266 3      EOL_TOKEN ]: : LF, Terminates the command line.
621 2267 3      ++
622 2268 3      Just return the token type directly.
623 2269 3      --
624 2270 3      .TOKEN_TYPE;
625 2271 3
626 2272 3
627 2273 3      [DIGIT_STR_TOKEN, : For branch operands where we interpret
628 2274 3      PERIOD_TOKEN]: : the operand as an absolute address.
629 2275 3      : We also handle operands of the
630 2276 3      : form '.' <sign> <number>, where
631 2277 3      : <sign> must be '+' or '-'.
632 2278 3
633 2279 4      BEGIN
634 2280 4      LOCAL
635 2281 4          BR_FLAG, NUMBER, SIGN;
636 2282 4
637 2283 4      ++
638 2284 4      In this case we build a token string consisting of a flag byte
639 2285 4      followed by the branch operand in a longword. Assume
640 2286 4      PC-relative type branching.
641 2287 4      --
642 2288 4      BR_FLAG = 1;
643 2289 4
644 2290 4      ++
645 2291 4      Handling the more-difficult syntax requires more work.
646 2292 4      --
647 2293 4      SIGN = 0;
648 2294 5      IF (.TOKEN_TYPE EQL PERIOD_TOKEN)
649 2295 4      THEN
650 2296 5          BEGIN
651 2297 5          BR_FLAG = 0;
652 2298 6          SIGN = (
653 2299 8              IF ((TOKEN_TYPE = MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC)
654 2300 7                  ) EQL PLUS_TOKEN)
655 2301 6              THEN
656 2302 6                  '+'
657 2303 6              ELSE
658 2304 7                  IF (.TOKEN_TYPE EQL MINUS_TOKEN )
659 2305 6                  THEN
660 2306 6                      '-'
661 2307 6                  ELSE
662 2308 7                      RETURN(BAD_TOKEN)
```

```
663 2309 5  
664 2310 5  
665 2311 5  
666 2312 5  
667 2313 5  
668 2314 6  
669 2315 5  
670 2316 5  
671 2317 4  
672 2318 4  
673 2319 4  
674 2320 4  
675 2321 4  
676 2322 4  
677 2323 5  
678 2324 4  
679 2325 4  
680 2326 4  
681 2327 4  
682 2328 4  
683 2329 4  
684 2330 3  
685 2331 3  
686 2332 3  
687 2333 3  
688 2334 3  
689 2335 4  
690 2336 4  
691 2337 4  
692 2338 4  
693 2339 4  
694 2340 4  
695 2341 4  
696 2342 5  
697 2343 4  
698 2344 5  
699 2345 5  
700 2346 6  
701 2347 5  
702 2348 5  
703 2349 5  
704 2350 5  
705 2351 5  
706 2352 5  
707 2353 4  
708 2354 4  
709 2355 5  
710 2356 4  
711 2357 4  
712 2358 4  
713 2359 4  
714 2360 4  
715 2361 4  
716 2362 4  
717 2363 4  
718 2364 5  
719 2365 4  
  
);  
  
++  
-- After the <sign>, a <number> is expected.  
--  
IF (MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC) NEQ DIGIT_STR_TOKEN)  
THEN  
    RETURN(BAD_TOKEN);  
END;  
  
++  
-- Pass back the flag and operand.  
--  
NUMBER = (.LEX_STG_DESC [DSC$A_POINTER]);  
IF (.SIGN EQL '=')  
THEN  
    NUMBER = - NUMBER;  
PASS BACK(BYTE_LENGTH, .BR_FLAG);  
RETURN PTR = .RETURN_PTR + BYTE_LENGTH;  
PASS BACK(LONG_LENGTH, .number);  
BRANCH_TOKEN  
END;  
! Pass back the token type  
  
[ALPHA_STR_TOKEN]:  
! For now, this must be a register reference,  
! or a displacement indicator ('B', 'W', 'L' or 'I').  
BEGIN  
    LOCAL  
        LEXEME_PTR,  
        CHAR;  
  
    LEXEME_PTR = CH$PTR (.LEX_STG_DESC [DSC$A_POINTER]);  
    CHAR = CH$RCHAR (.LEXEME_PTR);  
    IF ((REG_NUM = PAT$REG_MATCH(LEX_STG_DESC)) GEQ 0)  
    THEN  
        BEGIN  
            PASS_BACK(BYTE_LENGTH, .REG_NUM);  
            IF (.REG_NUM LEQ MAX_REG)  
            THEN  
                REGISTER_TOKEN  
            ELSE  
                BAD_TOKEN  
            END  
            ! Return the number of this register.  
            ! Valid register number  
            ! Invalid register number  
        ELSE IF (.LEX_STG_DESC [DSC$W_LENGTH] NEQ 1)  
        THEN  
            BAD_TOKEN  
        ELSE IF (.CHAR NEQ 'B') AND (.CHAR NEQ 'W') AND (.CHAR NEQ 'L') AND (.CHAR NEQ 'I')  
        THEN  
            ++  
            -- The string must be one of the valid  
            -- displacement CHARacters only.  
            --  
            BAD_TOKEN  
            ! Error. The string must be only 1 CHARACTER long.  
  
    ELSE IF (MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC) NEQ UP_ARROW_TOKEN)  
    THEN
```

ELSE

```
++
--
<Size> must be followed by '^' to produce <size indicator>.
--
BAD_TOKEN
++
Extract the following number and pass it back.
\This is where we would like to switch grammars
to take advantage of what PATCH already
knows about expression evaluation\.
```

```
++
--
BEGIN
++
If we are dealing with an immediate mode operand specified by the I^ notation, we
obliged to enforce the assembler syntax of the I^#<number or absolute expression>
notation (for consistency).
```

```
++
--
IF (.CHAR EQL 'I') THEN
    BEGIN
        IF (MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC) NEQ HASH_TOKEN)
            THEN
                ++
                --
                The following token must be #, if we are using I^.
```

```
++
--
                RETURN(BAD_TOKEN);
    END;
    IF (MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC) NEQ DIGIT_STR_TOKEN)
        THEN
            ++
            --
            The following token must be <number>.
```

```
++
--
            BAD_TOKEN
ELSE
    BEGIN
        ++
        --
        See what displacement mode was requested.
        MODE = 0;
        IF (.CHAR EQL 'B')
            THEN
                MODE = BYTE_LENGTH;
        IF (.CHAR EQL 'W')
            THEN
                MODE = WORD_LENGTH;
        IF (.CHAR EQL 'L')
            THEN
                MODE = LONG_LENGTH;
        IF (.CHAR EQL 'I')
            THEN
                BEGIN
                    PASS BACK(LONG_LENGTH, .(.LEX_STG_DESC [DSC$A_POINTER]));
                    RETURN (ABS_LIT_TOKEN);
                    END;
                IF (.MODE EQL 0)
                    THEN
                        RETURN(BAD_TOKEN);
```

```
! Immediate Mode operands must be ha
! autoincrement mode, with PC used a
! This is a special case of the # li
! Special Handling is called for, re
! for processing this type of immedi
```

```

      777 2423 6
      778 2424 6
      779 2425 6
      780 2426 6
      781 2427 6
      782 2428 6
      783 2429 6
      784 2430 6
      785 2431 6
      786 2432 6
      787 2433 6
      788 2434 5
      789 2435 3
      790 2436 3
      791 2437 3
      792 2438 3
      793 2439 3
      794 2440 3
      795 2441 3
      796 2442 3
      797 2443 3
      798 2444 3
      799 2445 3
      800 2446 3
      801 2447 4
      802 2448 3
      803 2449 3
      804 2450 3
      805 2451 4
      806 2452 3
      807 2453 3
      808 2454 3
      809 2455 3
      810 2456 3
      811 2457 3
      812 2458 4
      813 2459 3
      814 2460 3
      815 2461 3
      816 2462 4
      817 2463 4
      818 2464 4
      819 2465 4
      820 2466 4
      821 2467 5
      822 2468 7
      823 2469 6
      824 2470 6
      825 2471 6
      826 2472 6
      827 2473 5
      828 2474 4
      829 2475 4
      830 2476 4
      831 2477 5
      832 2478 5
      833 2479 5

      END;
      END

[OP_PAREN_TOKEN,
LSQUARE_TOKEN]:
      ++
      ! ( reg ), ETC.
      ! [ reg ], for INDEXing mode.
      --
      This case looks for an 'at register' reference or an indexing
      mode indicator. The reason why they are lumped together is
      simply due their similarity. The two tokens are in no
      other way related. The REGISTER number is passed back in the
      1st byte of LEX_BUFFER.
      --
      IF (MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC) NEQ ALPHA_STR_TOKEN)
      THEN
          BAD_TOKEN ! No parens or brackets around register
      ELSE
          IF ((REG_NUM = PAT$REG_MATCH(LEX_STG_DESC)) LSS 0)
          THEN
              BAD_TOKEN ! Invalid register name
          ELSE
              ++
              ! Make sure that the register is not PSL
              --
              IF (.REG_NUM GTR MAX_REG)
              THEN
                  BAD_TOKEN
              ELSE
                  BEGIN
                      ++
                      ! The next token must be a closing parenthesis
                      ! or square bracket. If it is not, error.
                      --
                      IF (MAR_GET_LEX(.INST_STG_DESC, LEX_STG_DESC)
                          NEQ ( IF (.TOKEN_TYPE EQL OP_PAREN_TOKEN)
                              THEN
                                  CL_PAREN_TOKEN
                              ELSE
                                  RSQUARE_TOKEN
                              ))
                      THEN
                          BAD_TOKEN
                      ELSE
                          BEGIN
                              ++
                              ! The operand was complete. Return the

```



```

834 2480 5
835 2481 5
836 2482 5
837 2483 5
838 2484 6
839 2485 5
840 2486 5
841 2487 5
842 2488 5
843 2489 5
844 2490 5
845 2491 6
846 2492 5
847 2493 5
848 2494 5
849 2495 5
850 2496 5
851 2497 3
852 2498 3
853 2499 3
854 2500 3 [OTHERWISE]:
855 2501 3
856 2502 3 BAD_TOKEN;
857 2503 3
858 2504 2 TES);
859 2505 2
860 2506 2 !++
861 2507 2 ! Update the counted string pointer which is being maintained elsewhere.
862 2508 2 ! The returned value is whatever was arrived at in the above CASE statement.
863 2509 2 !--
864 2510 2 RETURN (.TOKEN_TYPE);
865 2511 1 END;
```

```

! number and token type.
PASS_BACK(BYTE_LENGTH, .REG_NUM);
IF (.TOKEN_TYPE EQL OP_PAREN_TOKEN)
THEN
    AT_REG_TOKEN
ELSE
    !++
    ! You can't index off the PC
    !--
    IF (.REG_NUM EQL PC_REG)
    THEN
        BAD_TOKEN
    ELSE
        INDEXING_TOKEN
END
END;
! ERROR.
! Unrecognized token. Terminate the encoding.
```

```

01FC 0000 GET_NEXT_TOKEN:
58 00000000G EF 9E 00002 !WORD Save R2,R3,R4,R5,R6,R7,R8 : 2138
57 00000000V EF 9E 00009 MOVAB PAT$REG_MATCH, R8
5E BC AE 9E 00010 MOVAB MAR_GET_LEX, R7
56 08 AC D0 00014 MOVAB -68(SP), SP
03 12 00018 MOVL LEX_BUFFER, RETURN_PTR : 2221
56 6E 9E 0001A BNEQ 1$
38 AE B4 0001D 1$: MOVAB GARBAGE, RETURN_PTR : 2223
1C AE 9E 00020 CLRW LEX_STG_DESC : 2228
3C AE 1A B0 00025 MOVAB TOKEN_BUFFER, LEX_STG_DESC+4 : 2229
40 AE 9F 00029 MOVW #26, LEX_STG_DESC+8 : 2230
38 AE 9F 00029 PUSHAB LEX_STG_DESC : 2235
54 04 AC D0 0002C MOVL INST_STG_DESC, R4
54 DD 00030 PUSHL R4
67 02 FB 00032 CALLS #2, MAR_GET_LEX
52 50 90 00035 MOVB R0, TOKEN_TYPE
09 13 00038 BEQL 2$ : 2241
31 52 91 0003A CMPB TOKEN_TYPE, #49
04 1A 0003D BGTRU 2$
52 47 8F 90 0003F MOVB #71, TOKEN_TYPE : 2243
```

43	8F	52	91	00043	2\$:	CMPB	TOKEN_TYPE, #67	2248	
		1E	12	00047		BNEQ	4\$		
		38	AE	9F	00049	PUSHAB	LEX_STG_DESC	2252	
		54	DD	0004C		PUSHL	R4		
00000048	67	02	FB	0004E		CALLS	#2, MAR_GET_LEX		
	8F	50	D1	00051		CMPL	R0, #72		
		03	13	00058		BEQL	3\$		
		01D7	31	0005A		BRW	36\$		
	66	3C	BE	D0	0005D	3\$:	@LEX_STG_DESC+4, (RETURN_PTR)	2257	
	50	F8	8F	9A	00061	MOVZBL	#248, R0	2256	
		20	11	00065		BRB	6\$	2252	
	3D	52	91	00067	4\$:	CMPB	TOKEN_TYPE, #61	2262	
		18	13	0006A		BEQL	5\$		
41	8F	52	91	0006C		CMPB	TOKEN_TYPE, #65		
		12	13	00070		BEQL	5\$		
46	8F	52	91	00072		CMPB	TOKEN_TYPE, #70		
		0C	13	00076		BEQL	5\$		
4C	8F	52	91	00078		CMPB	TOKEN_TYPE, #76		
		06	13	0007C		BEQL	5\$		
63	8F	52	91	0007E		CMPB	TOKEN_TYPE, #99		
		05	12	00082		BNEQ	7\$		
	50	52	9A	00084	5\$:	MOVZBL	TOKEN_TYPE, R0	2270	
		65	11	00087	6\$:	BRB	14\$		
48	8F	52	91	00089	7\$:	CMPB	TOKEN_TYPE, #72	2273	
		06	13	0008D		BEQL	8\$		
48	8F	52	91	0008F		CMPB	TOKEN_TYPE, #75		
		58	12	00093		BNEQ	15\$		
	55	01	D0	00095	8\$:	MOV	#1, BR_FLAG	2288	
		53	D4	00098		CLRL	SIGN	2293	
48	8F	52	91	0009A		CMPB	TOKEN_TYPE, #75	2294	
		38	12	0009E		BNEQ	12\$		
		55	D4	000A0		CLRL	BR_FLAG	2297	
		38	AE	9F	000A2	PUSHAB	LEX_STG_DESC	2299	
		54	DD	000A5		PUSHL	R4		
	67	02	FB	000A7		CALLS	#2, MAR_GET_LEX		
0000004C	52	50	90	000AA		MOVB	R0, TOKEN_TYPE		
	8F	50	D1	000AD		CMPL	R0, #76	2300	
		05	12	000B4		BNEQ	9\$		
	53	28	D0	000B6		MOVL	#43, SIGN	2299	
		09	11	000B9		BRB	10\$		
46	8F	52	91	000BB	9\$:	CMPB	TOKEN_TYPE, #70	2304	
		12	12	000BF		BNEQ	11\$		
	53	2D	D0	000C1		MOVL	#45, SIGN		
		38	AE	9F	000C4	10\$:	PUSHAB	LEX_STG_DESC	2314
		54	DD	000C7		PUSHL	R4		
	67	02	FB	000C9		CALLS	#2, MAR_GET_LEX		
00000048	8F	50	D1	000CC		CMPL	R0, #72		
		03	13	000D3	11\$:	BEQL	12\$		
		00E5	31	000D5		BRW	27\$		
	50	3C	BE	D0	000D8	12\$:	@LEX_STG_DESC+4, NUMBER	2322	
	2D	53	D1	000DC		CMPL	SIGN, #45	2323	
		03	12	000DF		BNEQ	13\$		
	50	50	CE	000E1		MNEGL	NUMBER, NUMBER	2325	
	86	55	90	000E4	13\$:	MOVB	BR_FLAG, (RETURN_PTR)+	2326	
	66	50	D0	000E7		MOVL	NUMBER, (RETURN_PTR)	2328	
	50	F4	8F	9A	000EA	MOVZBL	#244, R0	2279	
		27	11	000EE	14\$:	BRB	17\$		

47	8F	52	91	000F0	15\$:	CMPB	TOKEN_TYPE, #71	2333	
		03	13	000F4		BEQL	16\$		
		00D7	31	000F6		BRW	29\$		
	50	3C	AE	D0	000F9	16\$:	MOVL	LEX_STG_DESC+4, LEXEME_PTR	2340
	55		60	9A	000FD		MOVZBL	(LEXEME_PTR), CHAR	2341
		38	AE	9F	00100		PUSHAB	LEX_STG_DESC	2342
	68		01	FB	00103		CALLS	#1, PAT\$REG_MATCH	
	53		50	D0	00106		MOVL	R0, REG_NUM	
			0F	19	00109		BLSS	18\$	
	66		53	90	0010B		MOVB	REG_NUM, (RETURN_PTR)	2345
	0F		53	D1	0010E		CMPL	REG_NUM, #15	2346
			72	14	00111		BGTR	22\$	
	50	F7	8F	9A	00113		MOVZBL	#247, R0	
		01	1E	31	00117	17\$:	BRW	37\$	
	01	38	AE	B1	0011A	18\$:	CMPW	LEX_STG_DESC, #1	2352
			65	12	0011E		BNEQ	22\$	
00000042	8F		55	D1	00120		CMPL	CHAR, #66	2355
			1B	13	00127		BEQL	19\$	
00000057	8F		55	D1	00129		CMPL	CHAR, #87	
			12	13	00130		BEQL	19\$	
0000004C	8F		55	D1	00132		CMPL	CHAR, #76	
			09	13	00139		BEQL	19\$	
00000049	8F		55	D1	0013B		CMPL	CHAR, #73	
			41	12	00142		BNEQ	22\$	
		38	AE	9F	00144	19\$:	PUSHAB	LEX_STG_DESC	2364
			54	DD	00147		PUSHL	R4	
	67		02	FB	00149		CALLS	#2, MAR_GET_LEX	
00000053	8F		50	D1	0014C		CMPL	R0, #83	
			03	13	00153		BEQL	20\$	
		0082	31	00155		BRW	30\$		
			53	D4	00158	20\$:	CLRL	R3	2383
00000049	8F		55	D1	0015A		CMPL	CHAR, #73	
			13	12	00161		BNEQ	21\$	
			53	D6	00163		INCL	R3	
		38	AE	9F	00165		PUSHAB	LEX_STG_DESC	2385
			54	DD	00168		PUSHL	R4	
	67		02	FB	0016A		CALLS	#2, MAR_GET_LEX	
00000043	8F		50	D1	0016D		CMPL	R0, #67	
			47	12	00174		BNEQ	27\$	
		38	AE	9F	00176	21\$:	PUSHAB	LEX_STG_DESC	2392
			54	DD	00179		PUSHL	R4	
	67		02	FB	0017B		CALLS	#2, MAR_GET_LEX	
00000048	8F		50	D1	0017E		CMPL	R0, #72	
			64	12	00185	22\$:	BNEQ	32\$	
			50	D4	00187		CLRL	MODE	2403
00000042	8F		55	D1	00189		CMPL	CHAR, #66	2404
			03	12	00190		BNEQ	23\$	
	50		01	D0	00192		MOVL	#1, MODE	2406
00000057	8F		55	D1	00195	23\$:	CMPL	CHAR, #87	2407
			03	12	0019C		BNEQ	24\$	
	50		02	D0	0019E		MOVL	#2, MODE	2409
0000004C	8F		55	D1	001A1	24\$:	CMPL	CHAR, #76	2410
			03	12	001A8		BNEQ	25\$	
	50		04	D0	001AA		MOVL	#4, MODE	2412
	09		53	E9	001AD	25\$:	BLBC	R3, 26\$	2413
	66	3C	BE	D0	001B0		MOVL	@LEX_STG_DESC+4, (RETURN_PTR)	2416
	50	F8	8F	9A	001B4		MOVZBL	#248, R0	2417

			04	001B8	RET			
		50	D5	001B9	26\$:	TSTL	MODE	2419
		05	12	001BB		BNEQ	28\$	
	50	F9	8F	9A	001BD	27\$:	MOVZBL	#249, R0 2421
			04	001C1		RET		
	86		50	90	001C2	28\$:	MOVB	MODE, (RETURN_PTR)+ 2429
	66	3C	BE	D0	001C5		MOVL	@LEX_STG_DESC+4, (RETURN_PTR) 2431
	50	00F1	C0	9E	001C9		MOVAB	241(R0), R0 2432
			68	11	001CE		BRB	37\$ 2342
45	8F		52	91	001D0	29\$:	CMPB	TOKEN_TYPE, #69 2438
			06	13	001D4		BEQL	31\$
49	8F		52	91	001D6		CMPB	TOKEN_TYPE, #73
			58	12	001DA	30\$:	BNEQ	36\$
		38	AE	9F	001DC	31\$:	PUSHAB	LEX_STG_DESC 2447
			54	DD	001DF		PUSHL	R4
	67		02	FB	001E1		CALLS	#2, MAR_GET_LEX
00000047	8F		50	D1	001E4		CMPL	R0, #71
			47	12	001EB	32\$:	BNEQ	36\$
		38	AE	9F	001ED		PUSHAB	LEX_STG_DESC 2451
	68		01	FB	001F0		CALLS	#1, PAT\$REG_MATCH
	53		50	D0	001F3		MOVL	R0, REG_NUM
			3C	19	001F6		BLSS	36\$
	0F		53	D1	001F8		CMPL	REG_NUM, #15 2458
			37	14	001FB		BGTR	36\$
		38	AE	9F	001FD		PUSHAB	LEX_STG_DESC 2467
			54	DD	00200		PUSHL	R4
	67		02	FB	00202		CALLS	#2, MAR_GET_LEX
			54	D4	00205		CLRL	R4
49	8F		52	91	00207		CMPB	TOKEN_TYPE, #73 2468
			07	12	0020B		BNEQ	33\$
			54	D6	0020D		INCL	R4
	51		3F	D0	0020F		MOVL	#63, R1
			04	11	00212		BRB	34\$
	51	50	8F	9A	00214	33\$:	MOVZBL	#80, R1
	51		50	D1	00218	34\$:	CMPL	R0, R1
			17	12	0021B		BNEQ	36\$
	66		53	90	0021D		MOVB	REG_NUM, (RETURN_PTR) 2482
	06		54	E9	00220		BLBC	R4, 35\$ 2484
	50	F6	8F	9A	00223		MOVZBL	#246, R0
			0F	11	00227		BRB	37\$
	0F		53	D1	00229	35\$:	CMPL	REG_NUM, #15 2491
			06	13	0022C		BEQL	36\$
	50		8F	9A	0022E		MOVZBL	#240, R0
			04	11	00232		BRB	37\$ 2447
	50	F9	8F	9A	00234	36\$:	MOVZBL	#249, R0 2500
	52		50	90	00238	37\$:	MOVB	R0, TOKEN_TYPE 2244
	50		52	9A	0023B		MOVZBL	TOKEN_TYPE, R0 2510
			04	0023E		RET		2511

; Routine Size: 575 bytes, Routine Base: _PAT\$CODE + 028C

```
867 2512 1 ROUTINE GET_LABEL ( INST_STG_DESC, OUT_PC, DEFINE_FLAG ) : NOVALUE =
868 2513 1
869 2514 1 ++
870 2515 1 Functional Description:
871 2516 1
872 2517 1 Scan the supposed instruction string to see if it has a label.
873 2518 1 If there is a label, then extract it and the following colon and
874 2519 1 update the instruction string descriptor, INST_STG_DESC. Then
875 2520 1 enter the label as an user defined symbol with a value equal to the
876 2521 1 current PC.
877 2522 1
878 2523 1 Calling Sequence:
879 2524 1
880 2525 1 GET_LABEL (INST_STG_DESC, OUT_PC, DEFINE_FLAG)
881 2526 1
882 2527 1 Inputs:
883 2528 1
884 2529 1 INST_STG_DESC -A pointer to the counted string which contains
885 2530 1 the instruction string we are working on.
886 2531 1 OUT_PC -PC at which the instruction will reside
887 2532 1 DEFINE_FLAG -Indicator if the label should be entered into the
888 2533 1 user's symbol table. TRUE=enter it; FALSE=don't enter it
889 2534 1
890 2535 1 Implicit Inputs:
891 2536 1
892 2537 1 none
893 2538 1
894 2539 1 Outputs:
895 2540 1
896 2541 1 none
897 2542 1
898 2543 1 Implicit Outputs:
899 2544 1
900 2545 1 The instruction string descriptor is updated to exclude the label and
901 2546 1 colon. The symbol is entered into the user defined symbol table.
902 2547 1
903 2548 1 Returned Value:
904 2549 1
905 2550 1 none
906 2551 1
907 2552 1 --
908 2553 1
909 2554 2 BEGIN
910 2555 2
911 2556 2 MAP
912 2557 2 INST_STG_DESC : REF BLOCK [, BYTE];
913 2558 2
914 2559 2 LOCAL
915 2560 2 CHAR,
916 2561 2 TOKEN_TYPE,
917 2562 2 INS_PTR : REF VECTOR[BYTE],
918 2563 2 INS_DESC : BLOCK[8,BYTE],
919 2564 2 LABEL_DESC : BLOCK[12, BYTE],
920 2565 2 LABEL_BUF : VECTOR[CHS_PER_LXEME, BYTE];
921 2566 2
922 2567 2 ++
923 2568 2 Initialize the local instruction string descriptor. This routine uses
```

Next character in instruction string
Type of token found
Pointer to first byte of instruction
Local copy of instruction string descriptor
String descriptor for current token
Buffer to hold label string

```

: 924      2569 2  ! its own copy in case there is no label on the instruction. Also initialize
: 925      2570 2  ! the label string descriptor.
: 926      2571 2  !--
: 927      2572 2  INS_DESC [DSC$W_LENGTH] = .INST_STG_DESC[DSC$W_LENGTH];
: 928      2573 2  INS_DESC [DSC$A_POINTER] = .INST_STG_DESC[DSC$A_POINTER];
: 929      2574 2  LABEL_DESC [DSC$W_LENGTH] = 0;
: 930      2575 2  LABEL_DESC [DSC$A_POINTER] = LABEL_BUF;
: 931      2576 2  LABEL_DESC [DSC$W_MAXLEN] = CHS_PER_LEXEME;
: 932      2577 2  TOKEN_TYPE = MAR_GET_LEX(INS_DESC, LABEL_DESC);
: 933      2578 3  IF ((.TOKEN_TYPE NEQ ALPHA_STR_TOKEN) AND (.TOKEN_TYPE NEQ DIGIT_STR_TOKEN))
: 934      2579 2  THEN
: 935      2580 2      RETURN;                                ! No label
: 936      2581 2  INS_PTR = CH$PTR(.INS_DESC[DSC$A_POINTER], 0);
: 937      2582 2  DO
: 938      2583 2      CHAR = CH$RCHAR_A(INS_PTR)
: 939      2584 2  UNTIL ((.CHAR NEQU ' ') AND (.CHAR NEQU ' '));
: 940      2585 3  IF (.CHAR NEQU ':')
: 941      2586 2  THEN
: 942      2587 2      RETURN;                                ! No label
: 943      2588 2
: 944      2589 2  !++
: 945      2590 2  ! Now update the return instruction descriptor.
: 946      2591 2  !--
: 947      2592 2  INST_STG_DESC[DSC$W_LENGTH] = .INST_STG_DESC[DSC$W_LENGTH] - (.INS_PTR - .INST_STG_DESC[DSC$A_POINTER]);
: 948      2593 2  INST_STG_DESC[DSC$A_POINTER] = .INS_PTR;
: 949      2594 2
: 950      2595 2  !++
: 951      2596 2  ! Now define the label as an user symbol with a value of the current PC.
: 952      2597 2  !--
: 953      2598 2  PAT$DEFINE_SYM(LABEL_DESC, .OUT_PC, FALSE);
: 954      2599 2  RETURN;
: 955      2600 1  END;
```

0004 00000 GET_LABEL:						
	5E	30	C2 00002	.WORD	Save R2	: 2512
	52	04	AC D0 00005	SUBL2	#48, SP	
28	AE	62	B0 00009	MOVL	INST_STG_DESC, R2	: 2572
2C	AE	04	A2 D0 0000D	MOVW	(R2), INS_DESC	
		1C	AE B4 00012	MOVL	4(R2), INS_DESC+4	: 2573
				CLRW	LABEL_DESC	: 2574
20	AE	6E	9E 00015	MOVAB	LABEL_BUF, LABEL_DESC+4	: 2575
24	AE	19	B0 00019	MOVW	#25, LABEL_DESC+8	: 2576
		1C	AE 9F 0001D	PUSHAB	LABEL_DESC	: 2577
		2C	AE 9F 00020	PUSHAB	INS_DESC	
00000000V	EF	02	FB 00023	CALLS	#2, MAR_GET_LEX	
00000047	8F	50	D1 0002A	CMPL	TOKEN_TYPE, #71	: 2578
		09	13 00031	BEQL	1\$	
00000048	8F	50	D1 00033	CMPL	TOKEN_TYPE, #72	
		31	12 0003A	BNEQ	3\$	
	51	2C	AE D0 0003C	1\$: MOVL	INS_DESC+4, INS_PTR	: 2581
	50		81 9A 00040	2\$: MOVZBL	(INS_PTR)+, CHAR	: 2583
	20		50 D1 00043	CMPL	CHAR, #32	: 2584
		F8	13 00046	BEQL	2\$	

PATENC
V04-000

C 13
16-Sep-1984 00:40:15
14-Sep-1984 12:52:32

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[PATCH.SRC]PATENC.B32;1
Page 29
(5)

		09	50	D1	00048	CMPL	CHAR, #9	:		
			F3	13	0004B	BEQL	2\$	:		
		3A	50	D1	0004D	CMPL	CHAR, #58	:	2585	
			1B	12	00050	BNEQ	3\$	:		
50	04	A2	51	C3	00052	SUBL3	INS_PTR, 4(R2), R0	:	2592	
		62	50	A0	00057	ADDW2	R0, -(R2)	:		
	04	A2	51	D0	0005A	MOVL	INS_PTR, 4(R2)	:	2593	
			7E	D4	0005E	CLRL	-(SP)	:	2598	
			08	AC	DD	00060	PUSHL	OUT_PC	:	
			24	AE	9F	00063	PUSHAB	LABEL_DESC	:	
00000000G		EF	03	FB	00066	CALLS	#3, PAT\$DEFINE_SYM	:		
			04	0006D	3\$:	RET		:	2600	

; Routine Size: 110 bytes, Routine Base: _PAT\$CODE + 04CB

```

: 957 2601 1 ROUTINE GET_OPCODE ( INST_STG_DESC, F_OUT_BYTE_PTR, OPINFO_PTR, OUT_PC ) =
: 958 2602 1
: 959 2603 1 ++
: 960 2604 1 Functional Description:
: 961 2605 1
: 962 2606 1 Scan the supposed instruction string to extract the opcode from it.
: 963 2607 1
: 964 2608 1 Calling Sequence:
: 965 2609 1
: 966 2610 1 GET_OPCODE (INST_STG_DESC, F_OUT_BYTE_PTR, OPINFO_PTR, OUT_PC)
: 967 2611 1
: 968 2612 1 Inputs:
: 969 2613 1
: 970 2614 1 INST_STG_DESC -A pointer to the counted string which contains
: 971 2615 1 the instruction string we are working on.
: 972 2616 1 F_OUT_BYTE_PTR -A pointer to where we should stuff the
: 973 2617 1 opcode if we find a valid one.
: 974 2618 1 OPINFO_PTR -A pointer to where we should stuff the address
: 975 2619 1 of the OPINFO record which we find as a match
: 976 2620 1 to the opcode we extract from the instruction string.
: 977 2621 1 OUT_PC -A pointer to a loc that is bumped for each byte
: 978 2622 1 placed in the instruction stream.
: 979 2623 1
: 980 2624 1 Implicit Inputs:
: 981 2625 1
: 982 2626 1 The OPINFO data structure which is described in PATINS.B32
: 983 2627 1
: 984 2628 1 Outputs:
: 985 2629 1
: 986 2630 1 None other than via the parameters given above.
: 987 2631 1
: 988 2632 1 Implicit Outputs:
: 989 2633 1
: 990 2634 1 None.
: 991 2635 1
: 992 2636 1 Returned Value:
: 993 2637 1
: 994 2638 1 A pointer to a counted string (built by this routine) which is the
: 995 2639 1 name of the opcode. If the opcode is invalid, an error message is
: 996 2640 1 SIGNALed and there is no return.
: 997 2641 1 --
: 998 2642 1
: 999 2643 2 BEGIN
: 1000 2644 2
: 1001 2645 2 MAP
: 1002 2646 2 OPINFO_PTR : REF VECTOR[.LONG], ! Pointer into the OPINFO table
: 1003 2647 2 OUT_PC : REF VECTOR[.LONG],
: 1004 2648 2 F_OUT_BYTE_PTR : REF VECTOR[.BYTE],
: 1005 2649 2 INST_STG_DESC : REF BLOCK[.BYTE];
: 1006 2650 2
: 1007 2651 2 BIND
: 1008 2652 2 OPINFO_RECORD = .OPINFO_PTR : REF BLOCK[OPTSIZE,BYTE],
: 1009 2653 2 OUT_BYTE_PTR = .F_OUT_BYTE_PTR : REF VECTOR;
: 1010 2654 2
: 1011 2655 2 OWN
: 1012 2656 2 OPCODE_NAME : VECTOR[8,BYTE]; ! Buffer to hold opcode string
: 1013 2657 2
```



```
1014 2658 2 LOCAL
1015 2659 2     TOKEN_TYPE,
1016 2660 2     OPCODE,                                ! Numeric Opcode.
1017 2661 2     OPCODE_STG_DESC : BLOCK [12, BYTE],
1018 2662 2     USER_OPCODE : VECTOR[ CHS_PER_LEXEME, BYTE];
1019 2663 2
1020 2664 2     !++
1021 2665 2     ! Make sure that the first token is a potential opcode.
1022 2666 2     !--
1023 2667 2 OPCODE_STG_DESC [DSC$W_LENGTH] = 0;
1024 2668 2 OPCODE_STG_DESC [DSC$A_POINTER] = USER_OPCODE;
1025 2669 2 OPCODE_STG_DESC [DSC$W_MAXLEN] = CHS_PER_LEXEME;
1026 2670 2 IF ((TOKEN_TYPE = MAR_GET_LEX(.INST_STG_DESC, OPCODE_STG_DESC)) NEQ ALPHA_STR_TOKEN)
1027 2671 2 THEN
1028 2672 2     IF (.TOKEN_TYPE GTR KEYWORD_RANGE)
1029 2673 2     THEN
1030 2674 2         SIGNAL(PATS_BADOPCODE+MSG$K_WARN, 2, .OPCODE_STG_DESC[DSC$W_LENGTH],
1031 2675 2             .OPCODE_STG_DESC[DSC$A_POINTER]); ! No return
1032 2676 2
1033 2677 2     !++
1034 2678 2     ! Look in the opcode table for a match to this string.
1035 2679 2     !--
1036 2680 2 IF ((OPCODE = OPCODE_MATCH( OPCODE_STG_DESC, .OPINFO_PTR)) LSS 0)
1037 2681 2 THEN
1038 2682 2     SIGNAL(PATS_BADOPCODE+MSG$K_WARN, 2, .OPCODE_STG_DESC[DSC$W_LENGTH], .OPCODE_STG_DESC[DSC$A_POINTER]); !
1039 2683 2
1040 2684 2     !++
1041 2685 2     ! Found it. Put the opcode byte into the instruction stream being built, unless
1042 2686 2     ! there is insufficient information to continue. This only happens when the
1043 2687 2     ! opcode is reserved, because PATCH does not know how many operands to expect.
1044 2688 2     ! If this is an assembler directive, then don't try to put an opcode into
1045 2689 2     ! the byte stream. Just return successfully.
1046 2690 2     !--
1047 2691 2 IF (.OPINFO_RECORD[OP_NUMOPS] EQL NOT_AN_OP)
1048 2692 2 THEN
1049 2693 2     SIGNAL(PATS_RESOPCODE+MSG$K_WARN, 2, .OPCODE_STG_DESC[DSC$W_LENGTH], .OPCODE_STG_DESC[DSC$A_POINTER]); !
1050 2694 2 IF (.OPINFO_RECORD[OP_NUMOPS] NEQ ASM_DIR_OP)
1051 2695 2 THEN
1052 2696 2     BEGIN
1053 2697 2     IF (.OPCODE AND XX'FF') EQL XX'FD'
1054 2698 2     THEN
1055 2699 2         BEGIN                                ! Output 1st byte of 2 byte OPcode
1056 2700 2         OUT_BYTE_PTR[0] = XX'FD';
1057 2701 2         OUT_BYTE_PTR = .OUT_BYTE_PTR + 1;
1058 2702 2         OUT_PC[0] = .OUT_PC[0] + 1;
1059 2703 2         OPCODE = .OPCODE ^ -8;
1060 2704 2         END;
1061 2705 2     OUT_BYTE_PTR[0] = .OPCODE;
1062 2706 2     OUT_BYTE_PTR = .OUT_BYTE_PTR + 1;
1063 2707 2     OUT_PC[0] = .OUT_PC[0] + 1;
1064 2708 2     END;
1065 2709 2
1066 2710 2     !++
1067 2711 2     ! Build a counted string which contains the name of the opcode for possible
1068 2712 2     ! use later in error reporting. Return a pointer to this string as the
1069 2713 2     ! result of this routine.
1070 2714 2     !--
```

```
: 1071      2715 2 CH$MOVE(.OPCO_STG_DESC[DSC$W_LENGTH], .OPCO_STG_DESC[DSC$A_POINTER],
: 1072      2716 2      OP$CODE_NAME[T]);
: 1073      2717 2 OP$CODE_NAME[0] = .OPCO_STG_DESC[DSC$W_LENGTH];
: 1074      2718 2 RETURN(OP$CODE_NAME);
: 1075      2719 1 END;
```

.PSECT _PAT\$OWN,NOEXE,2

00004 OP\$CODE_NAME:
.BLKB 8

.PSECT _PAT\$CODE,NOWRT,2

00FC 00000 GET_OPCODE:

	57	00000000'	EF	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7	2601
	56	00000000G	00	9E	00009	MOVAB	OP\$CODE_NAME, R7	
	5E		28	C2	00010	MOVAB	LIB\$SIGNAL, R6	
	53	08	AC	D0	00013	SUBL2	#40, SP	
		1C	AE	B4	00017	MOVL	F OUT BYTE PTR, R3	2653
20	AE		6E	9E	0001A	CLR	OPCO_STG_DESC	2667
24	AE		19	B0	0001E	MOVAB	USER_OPCODE, OPCO_STG_DESC+4	2668
		1C	AE	9F	00022	MOVW	#25, OPCO_STG_DESC+8	2669
		04	AC	DD	00025	PUSHAB	OPCO_STG_DESC	2670
00000000V	EF		02	FB	00028	PUSHL	INST_STG_DESC	
00000047	8F		50	D1	0002F	CALLS	#2, MAR_GET_LEX	
			17	13	00036	CMPL	TOKEN_TYPE, #71	
	31		50	D1	00038	BEQ	1\$	
			12	15	0003B	CMPL	TOKEN_TYPE, #49	2672
		20	AE	DD	0003D	BLEQ	1\$	
	7E	20	AE	3C	00040	PUSHL	OPCO_STG_DESC+4	2675
			02	DD	00044	MOVZWL	OPCO_STG_DESC, -(SP)	2674
		006D8208	8F	DD	00046	PUSHL	#2	
	66		04	FB	0004C	PUSHL	#7176712	
		0C	AC	DD	0004F	CALLS	#4, LIB\$SIGNAL	
		20	AE	9F	00052	PUSHL	OPINFO_PTR	2680
00000000V	EF		02	FB	00055	PUSHAB	OPCO_STG_DESC	
	54		50	D0	0005C	CALLS	#2, OP\$CODE_MATCH	
			12	18	0005F	MOVL	R0, OP\$CODE	
		20	AE	DD	00061	BGEQ	2\$	
	7E	20	AE	3C	00064	PUSHL	OPCO_STG_DESC+4	2682
			02	DD	00068	MOVZWL	OPCO_STG_DESC, -(SP)	
		006D8208	8F	DD	0006A	PUSHL	#2	
	66		04	FB	00070	PUSHL	#7176712	
	52	0C	BC	D0	00073	CALLS	#4, LIB\$SIGNAL	
FFFFFFFF	8F	04	A2	04	00	MOV	@OPINFO_PTR, R2	2691
				12	12	CMPL	#0, #4, -4(R2), #-1	
				AE	DD	BNEQ	3\$	
		20	AE	DD	00083	PUSHL	OPCO_STG_DESC+4	2693
	7E	20	AE	3C	00086	MOVZWL	OPCO_STG_DESC, -(SP)	
			02	DD	0008A	PUSHL	#2	
		006D8210	8F	DD	0008C	PUSHL	#7176720	
	66		04	FB	00092	CALLS	#4, LIB\$SIGNAL	
FFFFFFFFE	8F	04	A2	04	00	CMPL	#0, #4, 4(R2), #-2	2694

				1E 13 0009F	BEQL 5\$	:	
	FD	8F		54 91 000A1	CMPB OPCODE, #253	:	2697
				0F 12 000A5	BNEQ 4\$	:	
	00	B3	FD	8F 9A 000A7	MOVZBL #253, @0(R3)	:	2700
				63 D6 000AC	INCL (R3)	:	2701
			10	BC D6 000AE	INCL @OUT_PC	:	2702
54		54	F8	8F 78 000B1	ASHL #-8, OPCODE, OPCODE	:	2703
	00	B3		54 D0 000B6 4\$:	MOVL OPCODE, @0(R3)	:	2705
				63 D6 000BA	INCL (R3)	:	2706
			10	BC D6 000BC	INCL @OUT_PC	:	2707
01	A7	20	1C	AE 28 000BF 5\$:	MOVCL OPCODE_STG_DESC, @OPCODE_STG_DESC+4, -	:	2716
					OPCODE_NAME+1	:	
		67	1C	AE 90 000C6	MOVB OPCODE_STG_DESC, OPCODE_NAME	:	2717
		50		67 9E 000CA	MOVAB OPCODE_NAME, R0	:	2718
				04 000CD	RET	:	2719

; Routine Size: 206 bytes, Routine Base: _PAT\$CODE + 0539

```
1077 2720 1 ROUTINE GET_OPERAND ( INS_STG_DESC, OPRND_STG_DESC, USER_SYM_FLAG, OPR_CONTEXT ) =
1078 2721 1
1079 2722 1 ++
1080 2723 1 Functional Description:
1081 2724 1
1082 2725 1 This routine scans the input stream and extracts the next operand from
1083 2726 1 it. During this scan, all expressions and symbolic names are evaluated.
1084 2727 1 The resultant "operand" string is placed in the buffer pointed to by
1085 2728 1 OPRND_STG_DESC. This string can then be encoded by ENC_OPERAND into
1086 2729 1 a binary operand for the instruction.
1087 2730 1
1088 2731 1 Calling Sequence:
1089 2732 1
1090 2733 1 GET_OPERAND (INS_STG_DESC, OPRND_STG_DESC, USER_SYM_FLAG, OPR_CONTEXT)
1091 2734 1
1092 2735 1 Inputs:
1093 2736 1
1094 2737 1 INS_STG_DESC - Address of string descriptor for remaining operands of
1095 2738 1 instruction being encoded.
1096 2739 1 OPRND_STG_DESC - Address of string descriptor for buffer into which the
1097 2740 1 "pre-processed" operand string will be placed
1098 2741 1 USER_SYM_FLAG - Indicator whether or not to reduce user-defined symbols.
1099 2742 1 This feature is for command file output.
1100 2743 1 (TRUE=reduce, FALSE=don't reduce)
1101 2744 1 OPR_CONTEXT - Indicates the number of bytes for the operand used in
1102 2745 1 the current context. (Implements the I^ logic.)
1103 2746 1
1104 2747 1 Implicit Inputs:
1105 2748 1
1106 2749 1 PAT$GL_MOD_PTR - is used by the radix convert routine
1107 2750 1 to convert numeric input.
1108 2751 1
1109 2752 1 Outputs:
1110 2753 1
1111 2754 1 None.
1112 2755 1
1113 2756 1 Implicit Outputs:
1114 2757 1
1115 2758 1 OPRND_STG_DESC[DSC$W_LENGTH] will be set to the length of the operand.
1116 2759 1 The buffer pointed to by OPRND_STG_DESC[DSC$A_POINTER] will contain the
1117 2760 1 ASCII operand string.
1118 2761 1
1119 2762 1 Return Value:
1120 2763 1
1121 2764 1 OPR_FOUND (1) - if an operand is found.
1122 2765 1 NO_MORE_OPR (0) - if there are no more operands.
1123 2766 1 OPR_ERROR (2) - if there was an error in the operand.
1124 2767 1 OPR_FORW_REF (3) - if there was a forward reference to a symbol
1125 2768 1
1126 2769 1 Side Effects:
1127 2770 1
1128 2771 1 All errors in expressions or undefined symbols are SIGNALed, causing
1129 2772 1 an INFORMATIONAL message to be printed. Then an error code is returned
1130 2773 1 so that the routine that called PAT$INS_ENCODE will output the
1131 2774 1 instruction that could not be encoded.
1132 2775 1
1133 2776 1 Also, PAT$CP_OUT_STR and PAT$GL_BUF_SIZ are destroyed.
```

```
1134 2777 1 1
1135 2778 1 1--
1136 2779 1 1
1137 2780 2 BEGIN
1138 2781 2
1139 2782 2 MAP
1140 2783 2 INS STG_DESC : REF BLOCK [,BYTE],
1141 2784 2 OPRND_STG_DESC : REF BLOCK[,BYTE];
1142 2785 2
1143 2786 2 LABEL
1144 2787 2 DIGIT_CODE,
1145 2788 2 OPERATOR_CODE,
1146 2789 2 RANGLE_CODE,
1147 2790 2 ALPHA_CODE,
1148 2791 2 GET_PATHNAME;
1149 2792 2
1150 2793 2 BIND
1151 2794 2 EXP_STG = UPLIT BYTE (%ASCIC '^X!XL');
1152 2795 2
1153 2796 2 MACRO
1154 2797 2 EXP$L_VALUE = 0, 0, 32, 0%,
1155 2798 2 EXP$B_OPERATOR = 0, 32, 8, 1%;
1156 2799 2
1157 2800 2 LITERAL
1158 2801 2 EXP$C_SIZE = 5,
1159 2802 2 EXPR_STK_SIZ = EXP$C_SIZE * 20,
1160 2803 2 NEGATION_TOKEN = -1;
1161 2804 2
1162 2805 2 LOCAL
1163 2806 2 INS_START_PTR : REF VECTOR[,BYTE],
1164 2807 2 FLAG,
1165 2808 2 CUR_VALUE,
1166 2809 2 CUR_OPERATOR,
1167 2810 2 NUMBER_FLAG,
1168 2811 2 PREV_TOKEN,
1169 2812 2 TOKEN_TYPE,
1170 2813 2 LEX_STG_DESC : BLOCK[12,BYTE],
1171 2814 2 STACK_PTR : REF BLOCK[,BYTE],
1172 2815 2 EXPR_STACK : BLOCK[EXPR_STK_SIZ,BYTE],
1173 2816 2 LEX_BUF : VECTOR[CHS_PER_LEXEME,BYTE];
1174 2817 2
1175 2818 2 !++
1176 2819 2 ! First check that there is more input in the remaining operand string.
1177 2820 2 !--
1178 2821 3 IF (.INS_STG_DESC[DSC$W_LENGTH] EQL 0)
1179 2822 2 THEN
1180 2823 2 RETURN(NO_MORE_OPR);
1181 2824 2
1182 2825 2 !++
1183 2826 2 ! Initialize local variables before starting lexical scan.
1184 2827 2 !--
1185 2828 2 CUR_VALUE = 0;
1186 2829 2 CUR_OPERATOR = 0;
1187 2830 2 NUMBER_FLAG = FALSE;
1188 2831 2 TOKEN_TYPE = 0;
1189 2832 2 STACK_PTR = CH$PTR(EXPR_STACK, 0);
1190 2833 2 INS_START_PTR = CH$PTR(INS_STG_DESC[DSC$A_POINTER], 0);
```

! String descriptor for remaining operands
! String descriptor for returned operand

! Digit string token code
! Operator handling code
! Right angle bracket handling code
! Alpha_str_token handling code
! Code to get a pathname

! FAO control string to convert binary numbe

! EXP\$L_VALUE (expression value stacked)
! EXP\$B_OPERATOR (token_type for stacked ope

! Length of one expression stack entry
! Allow 20 entries on expression stack
! Token value for unary '-'

! Pointer to first byte of this operand
! Flag to exit operator select loop
! Current numeric value found
! Current operator to process
! Flag set if operand contained a numeric va
! Previous token_type value
! Encoded type of token found by PAT\$MAR_GET
! String descriptor for current lexeme
! Pointer to next free position on expressio
! Expression stack (grows from low address t
! Buffer to hold ascii lexeme string

```
1191 2834 2 BYTES_IN_OPRND = 0; ! Initialize to no bytes
1192 2835 2
1193 2836 2 !++
1194 2837 2 ! Zero the buffer which will hold the return operand.
1195 2838 2 !--
1196 2839 2 ZEROCOR (.OPRND_STG_DESC[DSC$A_POINTER], (.OPRND_STG_DESC[DSC$W_MAXLEN]/4));
1197 2840 2
1198 2841 2 !++
1199 2842 2 ! Now enter a loop to process the operand. There are two ways out of this loop.
1200 2843 2 ! The first is by SIGNALing an error. The second is by encountering a comma or
1201 2844 2 ! end of line. Some tokens may be passed directly into the operand string and
1202 2845 2 ! others indicate expressions or symbols. There is no check in this routine
1203 2846 2 ! of the validity of the resultant operand string.
1204 2847 2 !--
1205 2848 2 REPEAT
1206 2849 2 BEGIN
1207 2850 2 !++
1208 2851 2 ! Get the next token. The process this token based on its
1209 2852 2 ! encoded token_type.
1210 2853 2 !--
1211 2854 2 LEX_STG_DESC[DSC$W_LENGTH] = 0;
1212 2855 2 LEX_STG_DESC[DSC$A_POINTER] = LEX_BUF;
1213 2856 2 LEX_STG_DESC[DSC$W_MAXLEN] = CHS_PER_LEXEME;
1214 2857 2 PREV_TOKEN = .TOKEN_TYPE;
1215 2858 2 TOKEN_TYPE = MAR_GET_LEX(.INS_STG_DESC, LEX_STG_DESC);
1216 2859 2 SELECTONE .TOKEN_TYPE OF
1217 2860 2
1218 2861 2 SET
1219 2862 2
1220 2863 2 [CL PAREN_TOKEN, ! ')' as in (reg)
1221 2864 2 RSQUARE_TOKEN, ! ']' as in [reg]
1222 2865 2 HASH_TOKEN, ! '#' as in #literal
1223 2866 2 UP_ARROW_TOKEN, ! '^' as in B^,W^,...
1224 2867 2 PERIOD_TOKEN]: ! '.' as in .+offset, .-offset
1225 2868 2 BEGIN
1226 2869 2 !++
1227 2870 2 ! All of these tokens are passed directly into the resultant
1228 2871 2 ! operand string.
1229 2872 2 !--
1230 2873 2 ADD_LEX_T_OPRND( LEX_STG_DESC, .OPRND_STG_DESC);
1231 2874 2 END;
1232 2875 2
1233 2876 2
1234 2877 2 [OP PAREN_TOKEN, ! '(' as in (reg)
1235 2878 2 LSQUARE_TOKEN]: ! '[' as in [reg]
1236 2879 2 BEGIN
1237 2880 2 !++
1238 2881 2 ! These tokens signify the end of an expression, if there is
1239 2882 2 ! one. The expression and the token should be passed through.
1240 2883 2 !--
1241 2884 2 IF (.NUMBER_FLAG)
1242 2885 2 THEN
1243 2886 2 BEGIN
1244 2887 2 !++
1245 2888 2 ! Output the literal to the operand string. The case
1246 2889 2 ! here is number(reg) or number[reg].
1247 2890 2 !--
```

```
1248 2891 6      IF (.STACK_PTR NEQU EXPR_STACK)
1249 2892 5      THEN
1250 2893 6          BEGIN
1251 2894 6          SIGNAL(PATS_NOANGLE+MSG$K_INFO); ! Invalid expression - no closing '>'
1252 2895 6          RETURN(OPR_ERROR);
1253 2896 5          END;
1254 2897 5      PAT$GL_BUF_SIZ = 0;
1255 2898 5      PAT$CP_OUT_STR = CH$PTR(.OPRND_STG_DESC[DSC$A_POINTER],
1256 2899 5          .OPRND_STG_DESC[DSC$W_LENGTH]);
1257 2900 5      PAT$FAO_PUT(EXP_STG, .CUR_VALUE);
1258 2901 5      OPRND_STG_DESC[DSC$W_LENGTH] = .OPRND_STG_DESC[DSC$W_LENGTH] + .PAT$GL_BUF_SIZ;
1259 2902 5      NUMBER_FLAG = FALSE; ! Reset literal flag
1260 2903 5      CUR_VALUE = 0;
1261 2904 4      END;
1262 2905 4
1263 2906 4      ++
1264 2907 4      ! Now pass through the '(' or the '['.
1265 2908 4      --
1266 2909 4      ADD LEX T OPRND( LEX_STG_DESC, .OPRND_STG_DESC);
1267 2910 5      IF T.TOKEN_TYPE EQLU LSQUARE_TOKEN)
1268 2911 4      THEN
1269 2912 4          BYTES_IN_OPRND = .BYTES_IN_OPRND OR CONT_INDX_MASK;
1270 2913 3      END;
1271 2914 3
1272 2915 3
1273 2916 3      [AT SIGN_TOKEN,
1274 2917 3      PLUS_TOKEN,
1275 2918 3      MINUS_TOKEN,
1276 2919 3      SLASH_TOKEN,
1277 2920 3      ASTERISK_TOKEN];
1278 2921 3      OPERATOR_CODE:
1279 2922 4      BEGIN
1280 2923 4      ++
1281 2924 4      ! All operators are handled here. Later additions should be the
1282 2925 4      ! ampersand (&) for logical AND and exclamation point (!) for
1283 2926 4      ! logical OR. These are not currently passed through the
1284 2927 4      ! lexical scanner.
1285 2928 4      --
1286 2929 5      IF (.TOKEN_TYPE EQLU AT_SIGN_TOKEN) AND (.PREV_TOKEN EQL 0)
1287 2930 4      THEN
1288 2931 5          BEGIN
1289 2932 5          ++
1290 2933 5          ! There is no previous token therefore this must be the
1291 2934 5          ! @ (reg) case. Just pass the operator through to the
1292 2935 5          ! resultant operand string.
1293 2936 5          --
1294 2937 5          ADD LEX T OPRND( LEX_STG_DESC, .OPRND_STG_DESC);
1295 2938 5          LEAVE OPERATOR_CODE;
1296 2939 4          END;
1297 2940 4
1298 2941 4      IF ((.TOKEN_TYPE EQLU PLUS_TOKEN) OR (.TOKEN_TYPE EQLU MINUS_TOKEN)) AND
1299 2942 5      (.PREV_TOKEN EQLU PERIOD_TOKEN)
1300 2943 4      THEN
1301 2944 5          BEGIN
1302 2945 5          ++
1303 2946 5          ! This is the case of .+offset or .-offset; pass through
1304 2947 5          ! the operator to the resultant string.
```

```

--
ADD LEX T OPRND( LEX_STG_DESC, .OPRND_STG_DESC);
LEAVE OPERATOR_CODE;
END;

IF (.TOKEN_TYPE EQLU PLUS_TOKEN) AND (.PREV_TOKEN EQLU CL_PAREN_TOKEN)
THEN
    BEGIN
        ++
        This is the case (reg)+. Pass through the operator.
        --
        ADD LEX T OPRND( LEX_STG_DESC, .OPRND_STG_DESC);
        LEAVE OPERATOR_CODE;
        END;

IF (.TOKEN_TYPE EQLU MINUS_TOKEN)
THEN
    BEGIN
        ++
        Check for the case -(reg).
        --
        LOCAL
            NEXT_CHAR : BYTE,           ! Next character from operand stream
            INS_PTR : REF VECTOR[BYTE]; ! Pointer to next character in operand stream

        ++
        Find the next character in the operand stream that is
        not a space or a tab. This to check if the next
        character is a '('.
        --
        INS_PTR = CH$PTR(.INS_STG_DESC[DSC$A_POINTER],0);
        DO
            NEXT_CHAR = CH$RCHAR_A(INS_PTR)
        UNTIL ((.NEXT_CHAR NEQU ' ') AND (.NEXT_CHAR NEQU ' '));
        IF (.NEXT_CHAR EQLU '(')
        THEN
            BEGIN
                ++
                This was a case of -(reg). Pass through the '-'.
                --
                ADD LEX T OPRND( LEX_STG_DESC, .OPRND_STG_DESC);
                LEAVE OPERATOR_CODE;
                END;
            END;

        ++
        There should never be two successive operators unless the
        second is the unary negation operator. If the second is the
        negation operator, then stack it and the current value.
        Then set the current operator to be negation_token.
        --
        IF (.CUR_OPERATOR NEQ 0)
        THEN
            IF (.TOKEN_TYPE NEQU MINUS_TOKEN)
            THEN
                BEGIN
                    SIGNAL(PAT$MULTOPR+MSG$K_INFO); ! Two sequential operators found
                
```

```

: 1305 2948 5
: 1306 2949 5
: 1307 2950 5
: 1308 2951 4
: 1309 2952 4
: 1310 2953 5
: 1311 2954 4
: 1312 2955 5
: 1313 2956 5
: 1314 2957 5
: 1315 2958 5
: 1316 2959 5
: 1317 2960 5
: 1318 2961 4
: 1319 2962 4
: 1320 2963 5
: 1321 2964 4
: 1322 2965 5
: 1323 2966 5
: 1324 2967 5
: 1325 2968 5
: 1326 2969 5
: 1327 2970 5
: 1328 2971 5
: 1329 2972 5
: 1330 2973 5
: 1331 2974 5
: 1332 2975 5
: 1333 2976 5
: 1334 2977 5
: 1335 2978 5
: 1336 2979 5
: 1337 2980 5
: 1338 2981 5
: 1339 2982 6
: 1340 2983 5
: 1341 2984 6
: 1342 2985 6
: 1343 2986 6
: 1344 2987 6
: 1345 2988 6
: 1346 2989 6
: 1347 2990 5
: 1348 2991 4
: 1349 2992 4
: 1350 2993 4
: 1351 2994 4
: 1352 2995 4
: 1353 2996 4
: 1354 2997 4
: 1355 2998 4
: 1356 2999 5
: 1357 3000 4
: 1358 3001 5
: 1359 3002 4
: 1360 3003 5
: 1361 3004 5

```



```

: 1362      3005 5      RETURN(OPR_ERROR);
: 1363      3006 5      END
: 1364      3007 4      ELSE
: 1365      3008 5      BEGIN
: 1366      3009 6      IF ((.STACK_PTR + EXP$C_SIZE) GTRU (EXPR_STACK + EXPR_STK_SIZ))
: 1367      3010 5      THEN
: 1368      3011 6          BEGIN
: 1369      3012 6              SIGNAL(PAT$ EXPSTKOVN+MSG$K_INFO); ! Stack overflow
: 1370      3013 6              RETURN(OPR_ERROR);
: 1371      3014 5          END;
: 1372      3015 5
: 1373      3016 5      !++
: 1374      3017 5      ! Stack the current operator and value.
: 1375      3018 5      --
: 1376      3019 5      STACK_PTR[EXP$L_VALUE] = .CUR_VALUE;
: 1377      3020 5      STACK_PTR[EXP$B_OPERATOR] = .CUR_OPERATOR;
: 1378      3021 5      STACK_PTR = CH$PTR(.STACK_PTR, EXP$C_SIZE);
: 1379      3022 5      CUR_VALUE = 0;
: 1380      3023 5      CUR_OPERATOR = NEGATION_TOKEN;
: 1381      3024 5      LEAVE OPERATOR_CODE;
: 1382      3025 4      END;
: 1383      3026 4
: 1384      3027 4      !++
: 1385      3028 4      ! Set the new current operator and go get the next token.
: 1386      3029 4      --
: 1387      3030 4      CUR_OPERATOR = .TOKEN_TYPE;
: 1388      3031 3      END;
: 1389      3032 3
: 1390      3033 3
: 1391      3034 3      [LANGLE_TOKEN]: ! '<' encloses expression
: 1392      3035 4      BEGIN
: 1393      3036 4      !++
: 1394      3037 4      ! Check for expression stack overflow.
: 1395      3038 4      --
: 1396      3039 5      IF ((.STACK_PTR + EXP$C_SIZE) GTRU (EXPR_STACK + EXPR_STK_SIZ))
: 1397      3040 4      THEN
: 1398      3041 5          BEGIN
: 1399      3042 5              SIGNAL(PAT$ EXPSTKOVN+MSG$K_INFO); ! Stack overflow
: 1400      3043 5              RETURN(OPR_ERROR);
: 1401      3044 4          END;
: 1402      3045 4
: 1403      3046 4      !++
: 1404      3047 4      ! Put new entry on expression stack. Then reset the pointer
: 1405      3048 4      ! to the next unused entry. Push the current operator if there
: 1406      3049 4      ! is one, otherwise push a '<' as the operator.
: 1407      3050 4      --
: 1408      3051 4      STACK_PTR[EXP$L_VALUE] = .CUR_VALUE;
: 1409      3052 6      STACK_PTR[EXP$B_OPERATOR] = (IF (.CUR_OPERATOR NEQ 0)
: 1410      3053 5          THEN
: 1411      3054 5              .CUR_OPERATOR
: 1412      3055 5          ELSE
: 1413      3056 4              .TOKEN_TYPE);
: 1414      3057 4      STACK_PTR = CH$PTR(.STACK_PTR, EXP$C_SIZE);
: 1415      3058 4      CUR_VALUE = 0;
: 1416      3059 4      CUR_OPERATOR = 0;
: 1417      3060 3      END;
: 1418      3061 3
```

```
1419 3062 3
1420 3063 3
1421 3064 3
1422 3065 4
1423 3066 4
1424 3067 4
1425 3068 4
1426 3069 4
1427 3070 4
1428 3071 4
1429 3072 4
1430 3073 4
1431 3074 4
1432 3075 4
1433 3076 5
1434 3077 4
1435 3078 5
1436 3079 5
1437 3080 5
1438 3081 4
1439 3082 5
1440 3083 4
1441 3084 5
1442 3085 5
1443 3086 5
1444 3087 4
1445 3088 4
1446 3089 4
1447 3090 5
1448 3091 5
1449 3092 5
1450 3093 5
1451 3094 5
1452 3095 5
1453 3096 5
1454 3097 5
1455 3098 5
1456 3099 5
1457 3100 5 ! * FUTURE *
1458 3101 5 ! * FUTURE *
1459 3102 6
1460 3103 6
1461 3104 6
1462 3105 7
1463 3106 6
1464 3107 6
1465 3108 5
1466 3109 6
1467 3110 6
1468 3111 6
1469 3112 5
1470 3113 5
1471 3114 5
1472 3115 4
1473 3116 4
1474 3117 4
1475 3118 4

[ANGLE_TOKEN]:
RANGE_CODE:
BEGIN
++
The right angle bracket causes one entry to be taken off the
expression stack. This entry always immediately precedes the
next free entry pointed to by STACK_PTR. When an entry is
removed, the current value and the stacked value must be
combined using the stacked operator. If the operator is a
left angle bracket, then there were two consecutive left
angle brackets in the expression and there is no stacked value
to be combined.
--
IF (.STACK_PTR EQLU EXPR_STACK) ! Check for stack underflow
THEN
    BEGIN
        SIGNAL(PAT$ NOANGLE+MSG$K_INFO); ! Stack underflow
        RETURN(OPR_ERROR);
    END;
IF (.CUR_OPERATOR NEQU 0) OR (.PREV_TOKEN EQLU LANGLE_TOKEN) ! Check for missing operand
THEN
    BEGIN
        SIGNAL(PAT$ NOOPRND+MSG$K_INFO); ! Missing operand (got operator>)
        RETURN(OPR_ERROR);
    END;
STACK_PTR = CH$PTR(.STACK_PTR, -EXP$C_SIZE); ! Point to last entry
DO
BEGIN
FLAG = TRUE;
SELECTONE .STACK_PTR[EXP$B_OPERATOR] OF
    SET
    [PLUS_TOKEN]: CUR_VALUE = .STACK_PTR[EXP$S_VALUE] + .CUR_VALUE;
    [MINUS_TOKEN]: CUR_VALUE = .STACK_PTR[EXP$S_VALUE] - .CUR_VALUE;
    [ASTERISK_TOKEN]: CUR_VALUE = .STACK_PTR[EXP$S_VALUE] * .CUR_VALUE;
    [SLASH_TOKEN]: CUR_VALUE = .STACK_PTR[EXP$S_VALUE] / .CUR_VALUE;
    [AT_SIGN_TOKEN]: CUR_VALUE = .STACK_PTR[EXP$S_VALUE] ^ .CUR_VALUE;
    [LANGLE_TOKEN]: 0; ! No operation to perform
    [AMPERSAND_TOKEN]: CUR_VALUE = .STACK_PTR[EXP$S_VALUE] + .CUR_VALUE;
    [EXCLAM_PT_TOKEN]: CUR_VALUE = .STACK_PTR[EXP$S_VALUE] + .CUR_VALUE;
    [NEGATION_TOKEN]: BEGIN
        FLAG = FALSE;
        CUR_VALUE = -.CUR_VALUE;
        IF (.STACK_PTR NEQU EXPR_STACK) ! Check for stack underflow
        THEN
            STACK_PTR = CH$PTR(.STACK_PTR, -EXP$C_SIZE); ! Point
        END;
    [OTHERWISE]: BEGIN
        SIGNAL(PAT$ INVOPR+MSG$K_INFO); ! Unrecognized operand
        RETURN(OPR_ERROR);
    END;
    TES;
END
UNTIL (.FLAG);

CUR_OPERATOR = 0;
NUMBER_FLAG = TRUE;
```

```

: 1476      3119 3
: 1477      3120 3
: 1478      3121 3
: 1479      3122 3
: 1480      3123 4
: 1481      3124 5
: 1482      3125 4
: 1483      3126 5
: 1484      3127 4
: 1485      3128 4
: 1486      3129 5
: 1487      3130 5
: 1488      3131 6
: 1489      3132 5
: 1490      3133 5
: 1491      3134 5
: 1492      3135 5
: 1493      3136 5
: 1494      3137 5
: 1495      3138 5
: 1496      3139 5
: 1497      3140 5
: 1498      3141 5
: 1499      3142 5 ! * FUTURE *
: 1500      3143 5 ! * FUTURE *
: 1501      3144 6
: 1502      3145 6
: 1503      3146 6
: 1504      3147 7
: 1505      3148 6
: 1506      3149 7
: 1507      3150 7
: 1508      3151 7
: 1509      3152 7
: 1510      3153 7
: 1511      3154 6
: 1512      3155 5
: 1513      3156 6
: 1514      3157 6
: 1515      3158 6
: 1516      3159 5
: 1517      3160 5
: 1518      3161 5
: 1519      3162 4
: 1520      3163 4
: 1521      3164 4
: 1522      3165 3
: 1523      3166 3
: 1524      3167 3
: 1525      3168 3
: 1526      3169 3
: 1527      3170 4
: 1528      3171 4
: 1529      3172 4
: 1530      3173 4
: 1531      3174 4
: 1532      3175 4

      END;

[ DIGIT_STR_TOKEN]:                                ! numeric string
      BEGIN
      IF (.CUR_OPERATOR EQL 0)
      THEN
          CUR_VALUE = (.LEX_STG_DESC[DSC$A_POINTER])
      ELSE
          DO
          BEGIN
          FLAG = TRUE;
          IF (.CUR_OPERATOR EQL NEGATION_TOKEN)
          THEN
              CUR_VALUE = (.LEX_STG_DESC[DSC$A_POINTER]);
          SELECTONE .CUR_OPERATOR OF
              SET
          [PLUS_TOKEN]:  CUR_VALUE = .CUR_VALUE + (.LEX_STG_DESC[DSC$A_POINTER]);
          [MINUS_TOKEN]: CUR_VALUE = .CUR_VALUE - (.LEX_STG_DESC[DSC$A_POINTER]);
          [ASTERISK_TOKEN]: CUR_VALUE = .CUR_VALUE * (.LEX_STG_DESC[DSC$A_POINTER]);
          [SLASH_TOKEN]:  CUR_VALUE = .CUR_VALUE / (.LEX_STG_DESC[DSC$A_POINTER]);
          [AT_SIGN_TOKEN]: CUR_VALUE = .CUR_VALUE ^ (.LEX_STG_DESC[DSC$A_POINTER]);
          [LARGE_TOKEN]:  0;                                ! No operation to perform
          [AMPERSAND_TOKEN]: CUR_VALUE = .CUR_VALUE AND (.LEX_STG_DESC[DSC$A_POINTER]);
          [EXCLAM_PT_TOKEN]: CUR_VALUE = .CUR_VALUE OR (.LEX_STG_DESC[DSC$A_POINTER]);
          [NEGATION_TOKEN]: BEGIN
                          FLAG = FALSE;
                          CUR_VALUE = -.CUR_VALUE;
                          IF (.STACK_PTR NEQU EXPR_STACK) ! Check for stack underflow
                          THEN
                              BEGIN
                                  STACK_PTR = CH$PTR(.STACK_PTR, -EXP$C_SIZE); ! Point
                                  CUR_OPERATOR = .STACK_PTR[EXP$B_OPERATOR];
                                  .LEX_STG_DESC[DSC$A_POINTER] = .CUR_VALUE;
                                  CUR_VALUE = .STACK_PTR[EXP$L_VALUE];
                              END;
                          END;
          [OTHERWISE]: BEGIN
                          SIGNAL(PAT$ INVOPR+MSG$K_INFO); ! Unrecognized operand
                          RETURN(OPR_ERROR);
                      END;
          TES;
          END
          UNTIL (.FLAG);
          CUR_OPERATOR = 0;
          NUMBER_FLAG = TRUE;
          END;

[ EOL_TOKEN,
  COMMA_TOKEN]:                                ! End of operand string
      BEGIN
      ++
      ! Either of these tokens signals the end of the operand string.
      ! Check if there is a numeric value to be written to the
      ! resultant string. Check for expressions left on the stack.
      --
```

```
: 1533      3176 5      IF (.NUMBER_FLAG)
: 1534      3177 4      THEN
: 1535      3178 5          BEGIN
: 1536      3179 5              ++
: 1537      3180 5              | Output the current value as an ASCII string.
: 1538      3181 5              --
: 1539      3182 5              PAT$GL_BUF_SIZ = 0;
: 1540      3183 5              PAT$CP_OUT_STR = CH$PTR(.OPRND_STG_DESC[DSC$A_POINTER],
: 1541      3184 5                  .OPRND_STG_DESC[DSC$W_LENGTH]);
: 1542      3185 5              PAT$FAO_PUT(EXP_STG, .CUR_VALUE);
: 1543      3186 5              OPRND_STG_DESC[DSC$W_LENGTH] = .OPRND_STG_DESC[DSC$W_LENGTH] + .PAT$GL_BUF_SIZ;
: 1544      3187 4              END;
: 1545      3188 4
: 1546      3189 4          ++
: 1547      3190 4          | Check that there is not a missing operand, i.e., there exists
: 1548      3191 4          | a current operator.
: 1549      3192 4          --
: 1550      3193 5          IF (.CUR_OPERATOR NEQ 0)
: 1551      3194 4          THEN
: 1552      3195 5              BEGIN
: 1553      3196 5              SIGNAL(PAT$ NOOPRND+MSG$K_INFO);          ! Missing operand (got operator>)
: 1554      3197 5              RETURN(OPR_ERROR);
: 1555      3198 4              END;
: 1556      3199 4
: 1557      3200 4          ++
: 1558      3201 4          | Check that there was nothing left on the expression stack.
: 1559      3202 4          --
: 1560      3203 5          IF (.STACK_PTR NEQU EXPR_STACK)
: 1561      3204 4          THEN
: 1562      3205 5              BEGIN
: 1563      3206 5              SIGNAL(PAT$ NOANGLE+MSG$K_INFO);          ! Expression stack not empty
: 1564      3207 5              RETURN(OPR_ERROR);
: 1565      3208 4              END;
: 1566      3209 5          IF (.OPRND_STG_DESC[DSC$W_LENGTH] EQL 0)
: 1567      3210 4          THEN
: 1568      3211 5              RETURN(NO_MORE_OPR)          ! Got , or EOL_TOKEN and no operand
: 1569      3212 4          ELSE
: 1570      3213 4              RETURN(OPR_FOUND);
: 1571      3214 3          END;
: 1572      3215 3
: 1573      3216 3      [ALPHA STR TOKEN]:          ! Register, L^,B^,... or symbolic name
: 1574      3217 3      ALPHA_CODE:
: 1575      3218 3          BEGIN
: 1576      3219 4              ++
: 1577      3220 4              | First check if the string is a register. If so then pass
: 1578      3221 4              | it through.
: 1579      3222 4              --
: 1580      3223 4              LOCAL
: 1581      3224 4                  STATUS,
: 1582      3225 4                  SYMBOL_BUF : VECTOR[MAX BUF_SIZ,BYTE],
: 1583      3226 4                  SYMBOL_DESC : BLOCK[12,BYTE],
: 1584      3227 4                  NEXT_CHAR : BYTE,
: 1585      3228 4                  INS_PTR : REF VECTOR[,BYTE],
: 1586      3229 4                  REG_NUM,
: 1587      3230 4                  LEXEME_PTR,
: 1588      3231 4                  TEMP_VALUE,
: 1589      3232 4                  ! Return status from symbol definition
: 1589      3232 4                  ! Buffer to hold all of pathname
: 1589      3232 4                  ! String descriptor for pathname
: 1589      3232 4                  ! Next ASCII character in input stream
: 1589      3232 4                  ! Pointer to next character in input stream
: 1589      3232 4                  ! Number of register found
: 1589      3232 4                  ! Pointer to ASCII lexeme
: 1589      3232 4                  ! Value of pathname
```

1590 3233 4
1591 3234 4
1592 3235 4
1593 3236 5
1594 3237 4
1595 3238 5
1596 3239 5
1597 3240 5
1598 3241 5
1599 3242 5
1600 3243 5
1601 3244 4
1602 3245 4
1603 3246 4
1604 3247 4
1605 3248 4
1606 3249 4
1607 3250 4
1608 3251 4
1609 3252 4
1610 3253 5
1611 3254 4
1612 3255 4
1613 3256 4
1614 3257 4
1615 3258 4
1616 3259 5
1617 3260 4
1618 3261 5
1619 3262 5
1620 3263 5
1621 3264 6
1622 3265 5
1623 3266 6
1624 3267 6
1625 3268 6
1626 3269 6
1627 3270 6
1628 3271 6
1629 3272 7
1630 3273 6
1631 3274 6
1632 3275 7
1633 3276 6
1634 3277 6
1635 3278 7
1636 3279 6
1637 3280 6
1638 3281 7
1639 3282 6
1640 3283 7
1641 3284 7
1642 3285 7
1643 3286 6
1644 3287 6
1645 3288 5
1646 3289 4

```
CHAR : BYTE;                                ! First ASCII character in lexeme

IF ((REG_NUM = PAT$REG_MATCH(LEX_STG_DESC)) GEQ 0) AND
   (.REG_NUM LEQ MAX_REG)
THEN
    BEGIN
        ++
        Found a register. Pass it through.
        --
        ADD LEX_T OPRND( LEX_STG_DESC, .OPRND_STG_DESC);
        LEAVE ALPHA_CODE;
    END;

    ++
    It was not a register. Check if it is one of the following
    directives: L^, W^, B^, I^, or S^. If so, pass the token
    through.
    --
    LEXEME_PTR = CH$PTR(.LEX_STG_DESC[DSC$A_POINTER], 0);
    CHAR = CH$RCHAR(.LEXEME_PTR);
    IF (.LEX_STG_DESC[DSC$W_LENGTH] EQL 1)
    THEN
        IF (.CHAR EQLU 'B') OR
           (.CHAR EQLU 'W') OR
           (.CHAR EQLU 'L') OR
           (.CHAR EQLU 'I') OR
           (.CHAR EQLU 'S')
        THEN
            BEGIN
                INS_PTR = CH$PTR(.INS_STG_DESC[DSC$A_POINTER], 0);
                NEXT_CHAR = CH$RCHAR(.INS_PTR);
                IF (.NEXT_CHAR EQLU '^')
                THEN
                    BEGIN
                        LEX_BUF[1] = '^';
                        LEX_STG_DESC[DSC$W_LENGTH] = .LEX_STG_DESC[DSC$W_LENGTH] + 1;
                        INS_STG_DESC[DSC$W_LENGTH] = .INS_STG_DESC[DSC$W_LENGTH] - 1;
                        INS_STG_DESC[DSC$A_POINTER] = .INS_STG_DESC[DSC$A_POINTER] + 1;
                        ADD LEX_T OPRND(LEX_STG_DESC, .OPRND_STG_DESC);
                        IF (.CHAR EQLU 'B')
                        THEN
                            BYTES_IN_OPRND = .BYTES_IN_OPRND OR B_HAT_MASK;
                        IF (.CHAR EQLU 'W')
                        THEN
                            BYTES_IN_OPRND = .BYTES_IN_OPRND OR W_HAT_MASK;
                        IF (.CHAR EQLU 'L')
                        THEN
                            BYTES_IN_OPRND = .BYTES_IN_OPRND OR L_HAT_MASK;
                        IF (.CHAR EQLU 'I')
                        THEN
                            BEGIN
                                BYTES_IN_OPRND = .BYTES_IN_OPRND OR .OPR_CONTEXT;
                                PAT$G_CONTEXT[I_HAT_SEEN] = TRUE;
                            END;
                        LEAVE ALPHA_CODE;
                    END;
                END;
            END;
        END;
    END;
```

```

++
The alpha string must be a symbolic name.
Add the current token to the path name by calling PAT$BUILD_PATH.
Also build an ascii string of the pathname using ADD_LEX_T_OPRND
in case the symbol turns out to be undefined. This will be
used for forward branching in patch area.
--
SYMBOL_DESC[DSC$W_LENGTH] = 0;
SYMBOL_DESC[DSC$A_POINTER] = SYMBOL_BUF;
SYMBOL_DESC[DSC$W_MAXLEN] = MAX_BUF_SIZ;
ADD_LEX_T_OPRND(LEX_STG_DESC, SYMBOL_DESC);
IF NOT PAT$BUILD_PATH(LEX_STG_DESC, 0, FALSE)
THEN
    SIGNAL(PAT$_NOFREE)                ! Insufficient free memory
ELSE
    GET_PATHNAME:
    BEGIN
    ++
    ! Now loop to get the rest of the pathname (if any).
    --
    REPEAT
    BEGIN
    INS_PTR = CH$PTR(.INS_STG_DESC[DSC$A_POINTER], 0);
    DO
        NEXT_CHAR = CH$RCHAR_A(INS_PTR)
    UNTIL ((.NEXT_CHAR NEQU ' ') AND (.NEXT_CHAR NEQU ' '));
    IF (.NEXT_CHAR NEQU '\')
    THEN
        LEAVE GET_PATHNAME
    ELSE
        BEGIN
        ++
        ! Found a backslash. There must be another
        ! piece of the pathname.
        --
        LOCAL
            TEMP_TOKEN;                ! Temporary token type

        ++
        ! Now PATCH should be able to get the next two
        ! tokens (BACK_SLASH_TOKEN and ALPHA_STR_TOKEN).
        ! Any other token types are errors.
        --
        LEX_STG_DESC[DSC$W_LENGTH] = 0;
        LEX_STG_DESC[DSC$A_POINTER] = LEX_BUF;
        LEX_STG_DESC[DSC$W_MAXLEN] = CH$PER_LEXEME;
        IF (TEMP_TOKEN = MAR_GET_LEX(.INS_STG_DESC, LEX_STG_DESC)) NEQU BACKSLASH_T
        THEN
            BEGIN
            ! ***** THIS NEEDS AN OPERAND!!!!!!!!!! WHICH ONE?
            SIGNAL(PAT$_INVPATH+MSG$K_INFO), ! Invalid pathname
            RETURN(OPR_ERROR);
            END;
            ADD_LEX_T_OPRND(LEX_STG_DESC, SYMBOL_DESC);
            LEX_STG_DESC[DSC$W_LENGTH] = 0;
            LEX_STG_DESC[DSC$A_POINTER] = LEX_BUF;

```

1704 3347 7
1705 3348 7
1706 3349 7
1707 3350 8
1708 3351 8
1709 3352 8
1710 3353 8
1711 3354 8
1712 3355 7
1713 3356 7
1714 3357 7
1715 3358 7
1716 3359 7
1717 3360 6
1718 3361 5
1719 3362 4
1720 3363 4
1721 3364 4
1722 3365 4
1723 3366 4
1724 3367 4
1725 3368 4
1726 3369 4
1727 3370 4
1728 3371 4
1729 3372 5
1730 3373 4
1731 3374 5
1732 3375 5
1733 3376 5
1734 3377 5
1735 3378 5
1736 3379 5
1737 3380 5
1738 3381 5
1739 3382 5
1740 3383 5
1741 3384 5
1742 3385 4
1743 3386 5
1744 3387 4
1745 3388 5
1746 3389 5
1747 3390 5
1748 3391 5
1749 3392 5
1750 3393 5
1751 3394 5
1752 3395 5
1753 3396 5
1754 3397 5
1755 3398 4
1756 3399 4
1757 3400 4
1758 3401 5
1759 3402 4
1760 3403 5

```
LEX_STG_DESC[DSC$W_MAXLEN] = CHS_PER_LEXEME;
IF (TEMP_TOKEN = MAR_GET_LEX(.INS_STG_DESC, LEX_STG_DESC)) NEQU ALPHA_STR_TO
THEN
    BEGIN
        ! ***** THIS NEEDS AN OPERAND!!!!!! WHICH ONE?
        SIGNAL(PAT$ INVPATH+MSG$K_INFO); ! Invalid pathname
        RETURN(OPR_ERROR);
        ! ***** THIS NEEDS AN OPERAND!!!!!! WHICH ONE?
        END;
    IF NOT PAT$BUILD_PATH(LEX_STG_DESC, 0, FALSE)
    THEN
        SIGNAL(PAT$ NOFREE); ! Insufficient free memory
        ADD_LEX_T_OPRND(LEX_STG_DESC, SYMBOL_DESC);
    END;
END;
END;

!++
! Now the complete pathname has been acquired. Try to
! convert it into a numerical value. First search the current
! symbol table for a label. Then search the old label symbol
! table. Lastly, if the symbol is not defined then
! PAT$BUILD_PATH returns FALSE.
!--
STATUS = PAT$FIND_SYM(SYMBOL_DESC);
IF (.STATUS EQLA 0)
THEN
    BEGIN
        !++
        ! Remember the current symbol table but force
        ! PAT$FIND_SYM to use the old contents label table.
        !--
        LOCAL
            TEMP_SYMTB_PTR;
        TEMP_SYMTB_PTR = .PAT$GL_SYMTB_PTR;
        PAT$GL_SYMTB_PTR = .PAT$GL_OLDLABELS;
        STATUS = PAT$FIND_SYM(SYMBOL_DESC);
        PAT$GL_SYMTB_PTR = .TEMP_SYMTB_PTR;
        END;
    IF (.STATUS NEQA 0)
    THEN
        BEGIN
            !++
            ! Symbol was defined as a label. Get the value and
            ! then change STATUS from the table entry address to
            ! a code which would be returned by PAT$BUILD_PATH.
            !--
            TEMP_VALUE = .SYM_VALUE(.STATUS);
            STATUS = PAT$K_USER_DEF;
            PAT$DELETE_PATH();
            END
        ELSE
            STATUS = PAT$BUILD_PATH(0, TEMP_VALUE, FALSE);
    IF (NOT .STATUS) OR
        ((.STATUS EQLU PAT$K_USER_DEF) AND (NOT .USER_SYM_FLAG)) ! If undefined symbol or
        ! symbol is user-defined and not
    THEN
        BEGIN
```

1761 3404 5
1762 3405 5
1763 3406 5
1764 3407 5
1765 3408 5
1766 3409 5
1767 3410 5
1768 3411 5
1769 3412 5
1770 3413 5
1771 3414 5
1772 3415 5
1773 3416 6
1774 3417 6
1775 3418 6
1776 3419 6
1777 3420 6
1778 3421 6
1779 3422 6
1780 3423 7
1781 3424 6
1782 3425 6
1783 3426 6
1784 3427 5
1785 3428 5
1786 3429 5
1787 3430 5
1788 3431 5
1789 3432 4
1790 3433 4
1791 3434 4
1792 3435 4
1793 3436 4
1794 3437 5
1795 3438 4
1796 3439 4
1797 3440 4
1798 3441 4
1799 3442 5
1800 3443 5
1801 3444 6
1802 3445 5
1803 3446 5
1804 3447 5
1805 3448 5
1806 3449 5
1807 3450 5
1808 3451 5
1809 3452 5
1810 3453 5
1811 3454 5
1812 3455 5
1813 3456 5
1814 3457 6
1815 3458 6
1816 3459 6
1817 3460 7

```
++
Either the symbol was undefined and assumed to be a
forward reference or it is a user-defined symbol and
this is a reduction for command file output and thus
the symbol is not to be reduced. In the case of a
forward reference, the symbol will be defined later
in the current PATCH command. Since PATCH cannot
handle the operand now, set the operand string
descriptor to point to the un-reduced ascii string
in the instruction buffer.
--
DO
    BEGIN
    ++
    This loop finds the end of the operand.
    --
    LEX_STG_DESC[DSC$W_LENGTH] = 0;
    LEX_STG_DESC[DSC$A_POINTER] = LEX_BUF;
    TOKEN_TYPE = MAR_GET_LEX(.INS_STG_DESC, LEX_STG_DESC);
    IF (.TOKEN_TYPE EQLU LSQUARE_TOKEN)
    THEN
        BYTES_IN_OPRND = .BYTES_IN_OPRND OR CONT_INDX_MASK;
    END
    UNTIL ((.TOKEN_TYPE EQL EOL_TOKEN) OR (.TOKEN_TYPE EQL COMMA_TOKEN));
    OPRND_STG_DESC[DSC$W_LENGTH] = .INS_STG_DESC[DSC$A_POINTER]
    - .INS_START_PTR;
    OPRND_STG_DESC[DSC$A_POINTER] = CH$PTR(.INS_START_PTR, 0);
    RETURN(OPR_FORW_REF);
END;

++
Now that a value has been found, process it as if a
DIGIT_STR_TOKEN had been found.
--
IF (.CUR_OPERATOR EQL 0)
THEN
    CUR_VALUE = .TEMP_VALUE
ELSE
    DO
    BEGIN
    FLAG = TRUE;
    IF (.CUR_OPERATOR EQL NEGATION_TOKEN)
    THEN
        CUR_VALUE = .TEMP_VALUE;
    SELECTONE .CUR_OPERATOR OF
    SET
    [PLUS_TOKEN]: CUR_VALUE = .CUR_VALUE + .TEMP_VALUE;
    [MINUS_TOKEN]: CUR_VALUE = .CUR_VALUE - .TEMP_VALUE;
    [ASTERISK_TOKEN]: CUR_VALUE = .CUR_VALUE * .TEMP_VALUE;
    [SLASH_TOKEN]: CUR_VALUE = .CUR_VALUE / .TEMP_VALUE;
    [AT_SIGN_TOKEN]: CUR_VALUE = .CUR_VALUE ^ .TEMP_VALUE;
    [LARGE_TOKEN]: 0; ! No operation to perform
    [AMPERSAND_TOKEN]: CUR_VALUE = .CUR_VALUE AND .TEMP_VALUE;
    [EXCLAM_PT_TOKEN]: CUR_VALUE = .CUR_VALUE OR .TEMP_VALUE;
    [NEGATION_TOKEN]:
        BEGIN
        FLAG = FALSE;
        CUR_VALUE = -.CUR_VALUE;
        IF (.STACK_PTR NEQU EXP$STACK) ! Check for stack underflow
```



```

: 1818      3461  6
: 1819      3462  7
: 1820      3463  7
: 1821      3464  7
: 1822      3465  7
: 1823      3466  7
: 1824      3467  6
: 1825      3468  5
: 1826      3469  6
: 1827      3470  6
: 1828      3471  6
: 1829      3472  5
: 1830      3473  5
: 1831      3474  5
: 1832      3475  4
: 1833      3476  4
: 1834      3477  4
: 1835      3478  3
: 1836      3479  3
: 1837      3480  3
: 1838      3481  3
: 1839      3482  4
: 1840      3483  4
: 1841      3484  4
: 1842      3485  3
: 1843      3486  3
: 1844      3487  3
: 1845      3488  2
: 1846      3489  1
: INFO#212      L1:2839
: Null expression appears in value-required context

      THEN
      BEGIN
      STACK_PTR = CH$PTR(.STACK_PTR, -EXP$C_SIZE); ! Point
      CUR_OPERATOR = .STACK_PTR[EXP$B_OPERATOR];
      TEMP_VALUE = .CUR_VALUE;
      CUR_VALUE = .STACK_PTR[EXP$L_VALUE];
      END;
      END;
      [OTHERWISE]: BEGIN
      SIGNAL(PAT$ INVOPR+MSG$K_INFO); ! Unrecognized operand
      RETURN(OPR_ERROR);
      END;
      TES;
      END
      UNTIL (.FLAG);
      CUR_OPERATOR = 0;
      NUMBER_FLAG = TRUE;
      END;
      [OTHERWISE]: ! Illegal token type
      BEGIN
      SIGNAL(PAT$ INVOPR+MSG$K_INFO); ! Unrecognized operand
      RETURN(OPR_ERROR);
      END;
      TES;
      END; ! End of REPEAT loop
      END;
      END;

```

```

      .PSECT _PAT$PLIT,NOWRT,NOEXE,0
      4C 58 21 58 5E 05 00000 P.AAA: .ASCII <5>\^X!XL\
      EXP_STG= P.AAA
      .PSECT _PAT$CODE,NOWRT,2
      OFFC 00000 GET_OPERAND:
      SE FD50 CE 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 ; 2720
      5B 04 AC D0 00007 MOVAB -688(SP), SP
      6B B5 0000B MOVL INS_STG_DESC, R11 ; 2821
      03 12 0000D TSTW (R11)
      0669 31 0000F BNEQ 1$
      58 7C 00012 BRW 101$
      08 AE D4 00014 CLRQ CUR_VALUE ; 2828
      SA D4 00017 CLRL NUMBER_FLAG ; 2830
      56 90 AD 9E 00019 CLRL TOKEN_TYPE ; 2831
      6E 04 AB 9E 0001D MOVAB EXPR_STACK, STACK_PTR ; 2832
      10 AE 00 BE D0 00021 MOVAB 4(R11), (SP) ; 2833
      00000000' EF D4 00026 MOVL @0(SP), INS_START_PTR
      CLRL BYTES_IN_OPRND ; 2834

```

50

00

57	08	AC	D0	0002C	MOVL	OPRND_STG_DESC, R7	2839	
50	08	A7	3C	00030	MOVZWL	8(R7), R0		
50		04	C6	00034	DIVL2	#4, R0		
50		04	C4	00037	MULL2	#4, R0		
6E		00	2C	0003A	MOVCS	#0, (SP), #0, R0, @4(R7)		
	04	B7		0003F				
	F4	AD	B4	00041	2\$: CLRW	LEX_STG_DESC	2854	
F8	AD	FF74	CD	9E	00044	MOVAB	LEX_BUF, LEX_STG_DESC+4	2855
FC	AD		19	B0	0004A	MOVW	#25, LEX_STG_DESC+8	2856
OC	AE		5A	D0	0004E	MOVL	TOKEN_TYPE, PREV_TOKEN	2857
	F4	AD	9F	00052	PUSHAB	LEX_STG_DESC	2858	
		5B	DD	00055	PUSHL	R11		
00000000V	EF		02	FB	00057	CALLS	#2, MAR GET LEX	
	5A		50	D0	0005E	MOVL	R0, TOKEN_TYPE	
	3F		5A	D1	00061	CMPL	TOKEN_TYPE, #63	2863
			22	13	00064	BEQL	3\$	
00000043	8F		5A	D1	00066	CMPL	TOKEN_TYPE, #67	
			19	13	0006D	BEQL	3\$	
0000004B	8F		5A	D1	0006F	CMPL	TOKEN_TYPE, #75	
			10	13	00076	BEQL	3\$	
00000050	8F		5A	D1	00078	CMPL	TOKEN_TYPE, #80	
			07	13	0007F	BEQL	3\$	
00000053	8F		5A	D1	00081	CMPL	TOKEN_TYPE, #83	
			03	12	00088	3\$: BNEQ	4\$	
			0312	31	0008A	BRW	66\$	
00000045	8F		5A	D1	0008D	4\$: CMPL	TOKEN_TYPE, #69	2877
			09	13	00094	BEQL	5\$	
00000049	8F		5A	D1	00096	CMPL	TOKEN_TYPE, #73	
			62	12	0009D	BNEQ	10\$	
	3F	08	AE	E9	0009F	5\$: BLBC	NUMBER_FLAG, 8\$	2884
	50	90	AD	9E	000A3	MOVAB	EXPR_STACK, R0	2891
	50		56	D1	000A7	CMPL	STACK_PTR, R0	
			09	13	000AA	BEQL	7\$	
		006D81EB	8F	DD	000AC	6\$: PUSHL	#7176683	2894
			05BB	31	000B2	BRW	100\$	
		00000000G	EF	D4	000B5	7\$: CLRL	PAT\$GL_BUF_SIZ	2897
	50		67	3C	000BB	MOVZWL	(R7), R0	2899
00000000G	EF	04	B740	9E	000BE	MOVAB	@4(R7)[R0], PAT\$CP_OUT_STR	
			58	DD	000C7	PUSHL	CUR_VALUE	2900
		00000000'	EF	9F	000C9	PUSHAB	EXP_STG	
00000000G	EF		02	FB	000CF	CALLS	#2, PAT\$FAO PUT	
	67	00000000G	EF	A0	000D6	ADDW2	PAT\$GL_BUF_SIZ, (R7)	2901
		08	AE	D4	000DD	CLRL	NUMBER_FLAG	2902
			58	D4	000E0	CLRL	CUR_VALUE	2903
			57	DD	000E2	8\$: PUSHL	R7	2909
		F4	AD	9F	000E4	PUSHAB	LEX_STG_DESC	
00000000V	EF		02	FB	000E7	CALLS	#2, ADD_LEX_T_OPRND	
00000045	8F		5A	D1	000EE	CMPL	TOKEN_TYPE, #89	2910
			07	12	000F5	BNEQ	9\$	
00000000'	EF		08	88	000F7	BISB2	#8, BYTES_IN_OPRND	2912
			FF40	31	000FE	9\$: BRW	2\$	2859
	3C		5A	D1	00101	10\$: CMPL	TOKEN_TYPE, #60	2916
			05	19	00104	BLSS	11\$	
	3D		5A	D1	00106	CMPL	TOKEN_TYPE, #61	
			1E	15	00109	BLEQ	12\$	
00000046	8F		5A	D1	0010B	11\$: CMPL	TOKEN_TYPE, #70	
			15	13	00112	BEQL	12\$	

0000004C	8F		5A	D1	00114	CMPL	TOKEN_TYPE, #76		
			0C	13	0011B	BEQL	12\$		
00000052	8F		5A	D1	0011D	CMPL	TOKEN_TYPE, #82		
			03	13	00124	BEQL	12\$		
		008B	31	00126	BRW	23\$			
	3D		5A	D1	00129	CMPL	TOKEN_TYPE, #61	2929	
			05	12	0012C	BNEQ	13\$		
		0C	AE	D5	0012E	TSTL	PREV_TOKEN		
			48	13	00131	BEQL	19\$		
			52	D4	00133	CLRL	R2	2941	
0000004C	8F		5A	D1	00135	CMPL	TOKEN_TYPE, #76		
			04	12	0013C	BNEQ	14\$		
			52	D6	0013E	INCL	R2		
			09	11	00140	BRB	15\$		
00000046	8F		5A	D1	00142	CMPL	TOKEN_TYPE, #70		
			0A	12	00149	BNEQ	16\$		
0000004B	8F	0C	AE	D1	0014B	CMPL	PREV_TOKEN, #75	2942	
			26	13	00153	BEQL	19\$		
	06		52	E9	00155	BLBC	R2, 17\$	2953	
	3F	0C	AE	D1	00158	CMPL	PREV_TOKEN, #63		
			1D	13	0015C	BEQL	19\$		
00000046	8F		5A	D1	0015E	CMPL	TOKEN_TYPE, #70	2963	
			19	12	00165	BNEQ	20\$		
	51	00	BE	D0	00167	MOVL	@0(SP), INS_PTR	2978	
	50		81	90	0016B	MOVB	(INS_PTR)+, NEXT_CHAR	2980	
	20		50	91	0016E	CMPB	NEXT_CHAR, #32	2981	
			F8	13	00171	BEQL	18\$		
	09		50	91	00173	CMPB	NEXT_CHAR, #9		
			F3	13	00176	BEQL	18\$		
	28		50	91	00178	CMPB	NEXT_CHAR, #40	2982	
			03	12	0017B	BNEQ	20\$		
		021F	31	0017D	BRW	66\$			
			59	D5	00180	TSTL	CUR_OPERATOR	2999	
			28	13	00182	BEQL	22\$		
00000046	8F		5A	D1	00184	CMPL	TOKEN_TYPE, #70	3001	
			08	13	0018B	BEQL	21\$		
		006D81DB	8F	DD	0018D	PUSHL	#7176667	3004	
			3B	11	00193	BRB	25\$		
	51	05	A6	9E	00195	MOVAB	5(R6), R1	3009	
	50	F4	AD	9E	00199	MOVAB	EXPR_STACK+100, R0		
	50		51	D1	0019D	CMPL	R1, R0		
			28	1A	001A0	BGTRU	24\$		
	86		58	D0	001A2	MOVL	CUR_VALUE, (STACK_PTR)+	3019	
	86		59	90	001A5	MOVB	CUR_OPERATOR, (STACK_PTR)+	3020	
			58	D4	001AB	CLRL	CUR_VALUE	3022	
	59		01	CE	001AA	MNEGL	#1 - CUR_OPERATOR	3023	
			3B	11	001AD	BRB	29\$	3024	
	59		5A	D0	001AF	MOVL	TOKEN_TYPE, CUR_OPERATOR	3030	
			36	11	001B2	BRB	29\$	2916	
00000044	8F		5A	D1	001B4	CMPL	TOKEN_TYPE, #68	3034	
			30	12	001BB	BNEQ	30\$		
	51	05	A6	9E	001BD	MOVAB	5(R6), R1	3039	
	50	F4	AD	9E	001C1	MOVAB	EXPR_STACK+100, R0		
	50		51	D1	001C5	CMPL	R1, R0		
			08	1B	001C8	BLEQU	26\$		
		006D81E3	8F	DD	001CA	PUSHL	#7176675	3042	
			47	11	001D0	BRB	34\$		

66	58	D0	001D2	26\$:	MOVL	CUR_VALUE, (STACK_PTR)	3051
	59	D5	001D5		TSTL	CUR_OPERATOR	3052
	05	13	001D7		BEQL	27\$	
50	59	D0	001D9		MOVL	CUR_OPERATOR, R0	3054
	03	11	001DC		BRB	28\$	
50	5A	D0	001DE	27\$:	MOVL	TOKEN_TYPE, R0	3056
04	A6	50	90 001E1	28\$:	MOVB	R0, 4(STACK_PTR)	3052
56	05	C0	001E5		ADDL2	#5, STACK_PTR	3057
	58	7C	001E8		CLRQ	CUR_VALUE	3058
0000004F	8F	FE54	31 001EA	29\$:	BRW	2\$	2859
	5A	D1	001ED	30\$:	CMPL	TOKEN_TYPE, #79	3063
	03	13	001F4		BEQL	31\$	
	008A	31	001F6		BRW	44\$	
50	AD	9E	001F9	31\$:	MOVAB	EXPR_STACK, R0	3076
50	56	D1	001FD		CMPL	STACK_PTR, R0	
	03	12	00200		BNEQ	32\$	
	FEA7	31	00202		BRW	6\$	
	59	D5	00205	32\$:	TSTL	CUR_OPERATOR	3082
	0A	12	00207		BNEQ	33\$	
00000044	8F	0C	AE D1 00209		CMPL	PREV_TOKEN, #68	
		09	12 00211		BNEQ	35\$	
	006D81F3	8F	DD 00213	33\$:	PUSHL	#7176691	3085
	0454	31	00219	34\$:	BRW	100\$	
56	05	C2	0021C	35\$:	SUBL2	#5, STACK_PTR	3088
54	01	D0	0021F	36\$:	MOVL	#1, FLAG	3091
50	04	A6	98 00222		CVTBL	4(STACK_PTR), R0	3092
4C	8F	50	91 00226		CMPB	R0, #76	3094
	05	12	0022A		BNEQ	37\$	
58	58	66	C0 0022C		ADDL2	(STACK_PTR), CUR_VALUE	
	4D	11	0022F		BRB	43\$	
46	8F	50	91 00231	37\$:	CMPB	R0, #70	3095
	06	12	00235		BNEQ	38\$	
58	66	58	C3 00237		SUBL3	CUR_VALUE, (STACK_PTR), CUR_VALUE	
	41	11	0023B		BRB	43\$	
	3C	50	91 0023D	38\$:	CMPB	R0, #60	3096
	05	12	00240		BNEQ	39\$	
	58	66	C4 00242		MULL2	(STACK_PTR), CUR_VALUE	
	37	11	00245		BRB	43\$	
52	8F	50	91 00247	39\$:	CMPB	R0, #82	3097
	06	12	0024B		BNEQ	40\$	
58	66	58	C7 0024D		DIVL3	CUR_VALUE, (STACK_PTR), CUR_VALUE	
	2B	11	00251		BRB	43\$	
	3D	50	91 00253	40\$:	CMPB	R0, #61	3098
	06	12	00256		BNEQ	41\$	
58	66	58	78 00258		ASHL	CUR_VALUE, (STACK_PTR), CUR_VALUE	
	20	11	0025C		BRB	43\$	
44	8F	50	91 0025E	41\$:	CMPB	R0, #68	3099
	1A	13	00262		BEQL	43\$	
FF	8F	50	91 00264		CMPB	R0, #-1	3102
	03	13	00268		BEQL	42\$	
	03FD	31	0026A		BRW	99\$	
	54	D4	0026D	42\$:	CLRL	FLAG	3103
58	58	CE	0026F		MNEGL	CUR_VALUE, CUR_VALUE	3104
50	AD	9E	00272		MOVAB	EXPR_STACK, R0	3105
50	56	D1	00276		CMPL	STACK_PTR, R0	
	03	13	00279		BEQL	43\$	
56	05	C2	0027B		SUBL2	#5, STACK_PTR	3107

	9E		54	E9	0027E	43\$:	BLBC	FLAG, 36\$	3115
			14	11	00281		BRB	46\$	3117
00000048	8F		5A	D1	00283	44\$:	CMPL	TOKEN_TYPE, #72	3122
			03	13	0028A		BEQL	45\$	
		0095	31	0028C			BRW	58\$	
			59	D5	0028F	45\$:	TSTL	CUR_OPERATOR	3124
			07	12	00291		BNEQ	47\$	
	58	F8	BD	D0	00293		MOVL	@LEX_STG_DESC+4, CUR_VALUE	3126
		0084	31	00297	46\$:	BRW	56\$		
	54		01	D0	0029A	47\$:	MOVL	#1, FLAG	3130
FFFFFFFF	8F		59	D1	0029D		CMPL	CUR_OPERATOR, #-1	3131
			04	12	002A4		BNEQ	48\$	
	58	F8	BD	D0	002A6		MOVL	@LEX_STG_DESC+4, CUR_VALUE	3133
0000004C	8F		59	D1	002AA	48\$:	CMPL	CUR_OPERATOR, #76	3136
			06	12	002B1		BNEQ	49\$	
	58	F8	BD	C0	002B3		ADDL2	@LEX_STG_DESC+4, CUR_VALUE	
			62	11	002B7		BRB	55\$	
00000046	8F		59	D1	002B9	49\$:	CMPL	CUR_OPERATOR, #70	3137
			06	12	002C0		BNEQ	50\$	
	58	F8	BD	C2	002C2		SUBL2	@LEX_STG_DESC+4, CUR_VALUE	
			53	11	002C6		BRB	55\$	
	3C		59	D1	002C8	50\$:	CMPL	CUR_OPERATOR, #60	3138
			06	12	002CB		BNEQ	51\$	
	58	F8	BD	C4	002CD		MULL2	@LEX_STG_DESC+4, CUR_VALUE	
			48	11	002D1		BRB	55\$	
00000052	8F		59	D1	002D3	51\$:	CMPL	CUR_OPERATOR, #82	3139
			06	12	002DA		BNEQ	52\$	
	58	F8	BD	C6	002DC		DIVL2	@LEX_STG_DESC+4, CUR_VALUE	
			39	11	002E0		BRB	55\$	
	3D		59	D1	002E2	52\$:	CMPL	CUR_OPERATOR, #61	3140
			07	12	002E5		BNEQ	53\$	
58	58	F8	BD	78	002E7		ASHL	@LEX_STG_DESC+4, CUR_VALUE, CUR_VALUE	
			2D	11	002EC		BRB	55\$	
00000044	8F		59	D1	002EE	53\$:	CMPL	CUR_OPERATOR, #68	3141
			24	13	002F5		BEQL	55\$	
FFFFFFFF	8F		59	D1	002F7		CMPL	CUR_OPERATOR, #-1	3144
			03	13	002FE		BEQL	54\$	
		0367	31	00300		BRW	99\$		
			54	D4	00303	54\$:	CLRL	FLAG	3145
	58		58	CE	00305		MNEGL	CUR_VALUE, CUR_VALUE	3146
	50	90	AD	9E	00308		MOVAB	EXPR_STACK, R0	3147
	50		56	D1	0030C		CMPL	STACK_PTR, R0	
			0A	13	0030F		BEQL	55\$	
	59		76	98	00311		CVTBL	-(STACK_PTR), CUR_OPERATOR	3151
	F8		58	D0	00314		MOVL	CUR_VALUE, @LEX_STG_DESC+4	3152
			76	D0	00318		MOVL	-(STACK_PTR), CUR_VALUE	3153
	03		54	E9	0031B	55\$:	BLBC	FLAG, 57\$	3162
		0340	31	0031E	56\$:	BRW	98\$		
		FF76	31	00321	57\$:	BRW	47\$		
00000041	8F		5A	D1	00324	58\$:	CMPL	TOKEN_TYPE, #65	3168
			09	13	0032B		BEQL	59\$	
00000063	8F		5A	D1	0032D		CMPL	TOKEN_TYPE, #99	
			4A	12	00334		BNEQ	64\$	
	28	08	AE	E9	00336	59\$:	BLBC	NUMBER_FLAG, 60\$	3176
		00000000G	EF	D4	0033A		CLRL	PAT\$GL_BUF_SIZ	3182
	50		67	3C	00340		MOVZWL	(R7), R0	3184
00000000G	EF	04	B740	9E	00343		MOVAB	@4(R7)[R0], PAT\$CP_OUT_STR	

00000000G	EF	00000000'	58	DD	0034C	PUSHL	CUR_VALUE	3185
			EF	9F	0034E	PUSHAB	EXP_STG	
	67	00000000G	02	FB	00354	CALLS	#2, PAT\$FAO PUT	
			EF	A0	0035B	ADDW2	PAT\$GL_BUF_SIZ, (R7)	3186
			59	D5	00362	TSTL	CUR_OPERATOR	3193
			03	13	00364	BEQL	61\$	
			FEAA	31	00366	BRW	33\$	
	50	90	AD	9E	00369	MOVAB	EXPR_STACK, R0	3203
	50		56	D1	0036D	CMPL	STACK_PTR, R0	
			03	13	00370	BEQL	62\$	
			FD37	31	00372	BRW	6\$	
			67	B5	00375	TSTW	(R7)	3209
			03	12	00377	BNEQ	63\$	
			02FF	31	00379	BRW	101\$	
	50		01	D0	0037C	MOVL	#1, R0	3213
				04	0037F	RET		
00000047	8F		5A	D1	00380	CMPL	TOKEN_TYPE, #71	3217
			03	13	00387	BEQL	65\$	
			02DE	31	00389	BRW	99\$	
		F4	AD	9F	0038C	PUSHAB	LEX_STG_DESC	3235
00000000G	EF		01	FB	0038F	CALLS	#1, PAT\$REG_MATCH	
			50	D5	00396	TSTL	REG_NUM	
			14	19	00398	BLSS	67\$	
	0F		50	D1	0039A	CMPL	REG_NUM, #15	3236
			0F	14	0039D	BGTR	67\$	
			57	DD	0039F	PUSHL	R7	3242
		F4	AD	9F	003A1	PUSHAB	LEX_STG_DESC	
00000000V	EF		02	FB	003A4	CALLS	#2, ADD_LEX_T_OPRND	
		FC93	31	003AB	BRW	2\$		3243
	50	F8	AD	D0	003AE	MOVL	LEX_STG_DESC+4, LEXEME_PTR	3251
	52		60	90	003B2	MOVB	(LEXEME_PTR), CHAR	3252
	01	F4	AD	B1	003B5	CMPW	LEX_STG_DESC, #1	3253
			31	12	003B9	BNEQ	70\$	
			55	D4	003BB	CLRL	R5	3255
42	8F		52	91	003BD	CMPB	CHAR, #66	
			04	12	003C1	BNEQ	68\$	
			55	D6	003C3	INCL	R5	
			18	11	003C5	BRB	69\$	
57	8F		52	91	003C7	CMPB	CHAR, #87	3256
			12	13	003CB	BEQL	69\$	
4C	8F		52	91	003CD	CMPB	CHAR, #76	3257
			0C	13	003D1	BEQL	69\$	
49	8F		52	91	003D3	CMPB	CHAR, #73	3258
			06	13	003D7	BEQL	69\$	
53	8F		52	91	003D9	CMPB	CHAR, #83	3259
			65	12	003DD	BNEQ	75\$	
04	AE	00	BE	D0	003DF	MOVL	@0(SP), INS_PTR	3262
	53	04	BE	90	003E4	MOVB	@INS_PTR, NEXT_CHAR	3263
5E	8F		53	91	003E8	CMPB	NEXT_CHAR, #94	3264
			56	12	003EC	BNEQ	75\$	
FF75	CD	5E	8F	90	003EE	MOVB	#94, LEX_BUF+1	3267
		F4	AD	B6	003F4	INCW	LEX_STG_DESC	3268
			6B	B7	003F7	DECW	(R1T)	3269
		00	BE	D6	003F9	INCL	@0(SP)	3270
			57	DD	003FC	PUSHL	R7	3271
00000000V	EF	F4	AD	9F	003FE	PUSHAB	LEX_STG_DESC	
			02	FB	00401	CALLS	#2, ADD_LEX_T_OPRND	

00000000'	07		55	E9	00408	BLBC	R5, 71\$	3272
57	EF		01	88	0040B	BISB2	#1, BYTES-IN_OPRND	3274
	8F		52	91	00412	CMPB	CHAR, #87-	3275
			07	12	00416	BNEQ	72\$	
00000000'	EF		02	88	00418	BISB2	#2, BYTES-IN_OPRND	3277
4C	8F		52	91	0041F	CMPB	CHAR, #76-	3278
			07	12	00423	BNEQ	73\$	
00000000'	EF		04	88	00425	BISB2	#4, BYTES-IN_OPRND	3280
49	8F		52	91	0042C	CMPB	CHAR, #73-	3281
			0F	12	00430	BNEQ	74\$	
00000000'	EF	10	AC	C8	00432	BISL2	OPR_CONTEXT, BYTES_IN_OPRND	3284
00000000G	EF		04	88	0043A	BISB2	#4, -PAT\$GL_CONTEXT+3	3285
			FBFD	31	00441	BRW	Z\$	3287
		18	AE	B4	00444	CLRW	SYMBOL_DESC	3298
		24	AE	9E	00447	MOVAB	SYMBOL_BUF, SYMBOL_DESC+4	3299
1C	AE	0200	8F	B0	0044C	MOVW	#512, SYMBOL_DESC+8	3300
20	AE	18	AE	9F	00452	PUSHAB	SYMBOL_DESC	3301
		F4	AD	9F	00455	PUSHAB	LEX_STG_DESC	
00000000V	EF		02	FB	00458	CALLS	#2, -ADD_LEX_T_OPRND	
			7E	7C	0045F	CLRW	-(SP)	3302
		F4	AD	9F	00461	PUSHAB	LEX_STG_DESC	
00000000G	EF		03	FB	00464	CALLS	#3, -PAT\$BUILD_PATH	
10			50	E8	0046B	BLBS	R0, 77\$	
		006D8112	8F	DD	0046E	PUSHL	#7176466	3304
00000000G	00		01	FB	00474	CALLS	#1, LIB\$SIGNAL	
			00A4	31	0047B	BRW	82\$	
04	AE	00	BE	D0	0047E	MOVL	@0(SP), INS_PTR	3313
53		04	BE	90	00483	MOVW	@INS_PTR, NEXT_CHAR	3315
		04	AE	D6	00487	INCL	INS_PTR	
			53	91	0048A	CMPB	NEXT_CHAR, #32	3316
		20	F4	13	0048D	BEQL	78\$	
			53	91	0048F	CMPB	NEXT_CHAR, #9	
		09	EF	13	00492	BEQL	78\$	
			53	91	00494	CMPB	NEXT_CHAR, #92	3317
5C	8F		E1	12	00498	BNEQ	76\$	
		F4	AD	B4	0049A	CLRW	LEX_STG_DESC	3334
		FF74	CD	9E	0049D	MOVAB	LEX_BUF, LEX_STG_DESC+4	3335
F8	AD		19	B0	004A3	MOVW	#25, LEX_STG_DESC+8	3336
FC	AD		AD	9F	004A7	PUSHAB	LEX_STG_DESC	3337
		F4	5B	DD	004AA	PUSHL	R11	
00000000V	EF		02	FB	004AC	CALLS	#2, MAR_GET_LEX	
	55		50	D0	004B3	MOVL	R0, TEMP_TOKEN	
	3E		55	D1	004B6	CMPL	TEMP_TOKEN, #62	
			32	12	004B9	BNEQ	79\$	
		18	AE	9F	004BB	PUSHAB	SYMBOL_DESC	3344
		F4	AD	9F	004BE	PUSHAB	LEX_STG_DESC	
00000000V	EF		02	FB	004C1	CALLS	#2, -ADD_LEX_T_OPRND	
		F4	AD	B4	004C8	CLRW	LEX_STG_DESC	3345
F8	AD	FF74	CD	9E	004CB	MOVAB	LEX_BUF, LEX_STG_DESC+4	3346
FC	AD		19	B0	004D1	MOVW	#25, LEX_STG_DESC+8	3347
		F4	AD	9F	004D5	PUSHAB	LEX_STG_DESC	3348
			5B	DD	004D8	PUSHL	R11	
00000000V	EF		02	FB	004DA	CALLS	#2, MAR_GET_LEX	
	55		50	D0	004E1	MOVL	R0, TEMP_TOKEN	
00000047	8F		55	D1	004E4	CMPL	TEMP_TOKEN, #71	
			09	13	004EB	BEQL	80\$	
		006D8203	8F	DD	004ED	PUSHL	#7176707	3352

			017A	31	004F3	BRW	100\$		
			7E	7C	004F6	80\$:	CLRQ	-(SP)	3356
		F4	AD	9F	004F8	PUSHAB	LEX_STG_DESC		
00000000G	EF		03	FB	004FB	CALLS	#3, PAT\$BUILD_PATH		
	0D		50	E8	00502	BLBS	R0, 81\$		
		006D8112	8F	DD	00505	PUSHL	#7176466	3358	
00000000G	00		01	FB	0050B	CALLS	#1, LIB\$SIGNAL		
		18	AE	9F	00512	81\$:	PUSHAB	SYMBOL_DESC	3359
		F4	AD	9F	00515	PUSHAB	LEX_STG_DESC		
00000000V	EF		02	FB	00518	CALLS	#2, ADD_LEX_T_OPRND		
			FF5C	31	0051F	BRW	77\$	3307	
		18	AE	9F	00522	82\$:	PUSHAB	SYMBOL_DESC	3371
00000000G	EF		01	FB	00525	CALLS	#1, PAT\$FIND_SYM		
	53		50	D0	0052C	MOVL	R0, STATUS		
			26	12	0052F	BNEQ	83\$	3372	
	55	00000000G	EF	D0	00531	MOVL	PAT\$GL_SYMTBPTR, TEMP_SYMTB_PTR	3381	
00000000G	EF	00000000G	EF	D0	00538	MOVL	PAT\$GL_OLDLABLS, PAT\$GL_SYMTBPTR	3382	
		18	AE	9F	00543	PUSHAB	SYMBOL_DESC	3383	
00000000G	EF		01	FB	00546	CALLS	#1, PAT\$FIND_SYM		
	53		50	D0	0054D	MOVL	R0, STATUS		
00000000G	EF		55	D0	00550	MOVL	TEMP_SYMTB_PTR, PAT\$GL_SYMTBPTR	3384	
			53	D5	00557	83\$:	TSTL	STATUS	3386
			11	13	00559	BEQL	84\$		
14	AE	08	A3	D0	0055B	MOVL	8(STATUS), TEMP_VALUE	3394	
	53		03	D0	00560	MOVL	#3, STATUS	3395	
00000000G	EF		00	FB	00563	CALLS	#0, PAT\$DELETE_PATH	3396	
			11	11	0056A	BRB	85\$	3386	
			7E	D4	0056C	84\$:	CLRL	-(SP)	3399
		18	AE	9F	0056E	PUSHAB	TEMP_VALUE		
			7E	D4	00571	CLRL	-(SP)		
00000000G	EF		03	FB	00573	CALLS	#3, PAT\$BUILD_PATH		
	53		50	D0	0057A	MOVL	R0, STATUS		
	09		53	E9	0057D	85\$:	BLBC	STATUS, 86\$	3400
	03		53	D1	00580	CMPL	STATUS, #3	3401	
			4D	12	00583	BNEQ	89\$		
	49	0C	AC	E8	00585	BLBS	USER_SYM_FLAG, 89\$		
		F4	AD	B4	00589	86\$:	CLRW	LEX_STG_DESC	3420
F8	AD	FF74	CD	9E	0058C	MOVAB	LEX_BUF, LEX_STG_DESC+4	3421	
		F4	AD	9F	00592	PUSHAB	LEX_STG_DESC	3422	
			5B	DD	00595	PUSHL	R11		
00000000V	EF		02	FB	00597	CALLS	#2, MAR_GET_LEX		
	5A		50	D0	0059E	MOVL	R0, TOKEN_TYPE		
00000045	8F		5A	D1	005A1	CMPL	TOKEN_TYPE, #69	3423	
			07	12	005A8	BNEQ	87\$		
00000000'	EF		08	88	005AA	87\$:	BISB2	#8, BYTES_IN_OPRND	3425
00000063	8F		5A	D1	005B1	CMPL	TOKEN_TYPE, #99	3427	
			09	13	005B8	BEQL	88\$		
00000041	8F		5A	D1	005BA	CMPL	TOKEN_TYPE, #65		
			C6	12	005C1	BNEQ	86\$		
67	00	BE	10	AE	A3	88\$:	SUBW3	INS_START_PTR, @0(SP), (R7)	3429
	04	A7	10	AE	D0	MOVL	INS_START_PTR, 4(R7)	3430	
		50	03	D0	005CE	MOVL	#3, R0	3431	
				04	005D1	RET			
			59	D5	005D2	89\$:	TSTL	CUR_OPERATOR	3437
			07	12	005D4	BNEQ	90\$		
	58		14	AE	D0	MOVL	TEMP_VALUE, CUR_VALUE	3439	
			0084	31	005DA	BRW	98\$		

FFFFFFF	54		01	D0	005DD	90\$:	MOVL	#1, FLAG	:	3443
	8F		59	D1	005E0		CMPL	CUR_OPERATOR, #-1	:	3444
			04	12	005E7		BNEQ	91\$	:	
	58	14	AE	D0	005E9		MOVL	TEMP_VALUE, CUR_VALUE	:	3446
0000004C	8F		59	D1	005ED	91\$:	CMPL	CUR_OPERATOR, #76	:	3449
			06	12	005F4		BNEQ	92\$	:	
	58	14	AE	C0	005F6		ADDL2	TEMP_VALUE, CUR_VALUE	:	
			5F	11	005FA		BRB	97\$	:	
00000046	8F		59	D1	005FC	92\$:	CMPL	CUR_OPERATOR, #70	:	3450
			06	12	00603		BNEQ	93\$	:	
	58	14	AE	C2	00605		SUBL2	TEMP_VALUE, CUR_VALUE	:	
			50	11	00609		BRB	97\$	:	
	3C		59	D1	0060B	93\$:	CMPL	CUR_OPERATOR, #60	:	3451
			06	12	0060E		BNEQ	94\$	:	
	58	14	AE	C4	00610		MULL2	TEMP_VALUE, CUR_VALUE	:	
			45	11	00614		BRB	97\$	:	
00000052	8F		59	D1	00616	94\$:	CMPL	CUR_OPERATOR, #82	:	3452
			06	12	0061D		BNEQ	95\$	:	
	58	14	AE	C6	0061F		DIVL2	TEMP_VALUE, CUR_VALUE	:	
			36	11	00623		BRB	97\$	:	
	3D		59	D1	00625	95\$:	CMPL	CUR_OPERATOR, #61	:	3453
			07	12	00628		BNEQ	96\$	:	
58	58	14	AE	78	0062A		ASHL	TEMP_VALUE, CUR_VALUE, CUR_VALUE	:	
			2A	11	0062F		BRB	97\$	:	
00000044	8F		59	D1	00631	96\$:	CMPL	CUR_OPERATOR, #68	:	3454
			21	13	00638		BEQL	97\$	:	
FFFFFFF	8F		59	D1	0063A		CMPL	CUR_OPERATOR, #-1	:	3457
			27	12	00641		BNEQ	99\$	:	
			54	D4	00643		CLRL	FLAG	:	3458
	58		58	CE	00645		MNEGL	CUR_VALUE, CUR_VALUE	:	3459
	50	90	AD	9E	00648		MOVAB	EXPR_STACK, R0	:	3460
	50		56	D1	0064C		CMPL	STACK_PTR, R0	:	
			0A	13	0064F		BEQL	97\$	:	
	59		76	98	00651		CVTBL	-(STACK_PTR), CUR_OPERATOR	:	3464
14	AE		58	D0	00654		MOVL	CUR_VALUE, TEMP_VALUE	:	3465
	58		76	D0	00658		MOVL	-(STACK_PTR), CUR_VALUE	:	3466
	03		54	E8	0065B	97\$:	BLBS	FLAG, 98\$	:	3475
			FF7C	31	0065E		BRW	90\$	:	
			59	D4	00661	98\$:	CLRL	CUR_OPERATOR	:	3476
08	AE		01	D0	00663		MOVL	#1, NUMBER_FLAG	:	3477
			F9D7	31	00667		BRW	2\$	:	3217
		006D81FB	8F	DD	0066A	99\$:	PUSHL	#7176699	:	3483
00000000G	00		01	FB	00670	100\$:	CALLS	#1, LIB\$SIGNAL	:	
	50		02	D0	00677		MOVL	#2, R0	:	3484
			04	0067A			RET		:	
			50	D4	0067B	101\$:	CLRL	R0	:	3489
			04	0067D			RET		:	

; Routine Size: 1662 bytes, Routine Base: _PAT\$CODE + 0607

```

: 1848 3490 1 ROUTINE ENC_OPERAND ( INST_STG_DESC, OUT_BYTE_STREAM, PC_REL_CONTEXT, BRANCH_SIZE, OUT_PC_PTR, OPINFO_PTR )
: 1849 3491 1
: 1850 3492 1
: 1851 3493 1 ++
: 1852 3494 1 Functional Description:
: 1853 3495 1 Scan (parse, whatever) the string that supposedly
: 1854 3496 1 represents one operand reference, and come up
: 1855 3497 1 with the machine code representation for it.
: 1856 3498 1
: 1857 3499 1 Inputs:
: 1858 3500 1
: 1859 3501 1 INST_STG_DESC -A pointer to the counted string that
: 1860 3502 1 contains the operand reference.
: 1861 3503 1 OUT_BYTE_STREAM -A pointer to the output byte stream pointer
: 1862 3504 1 that is being maintained by the routine
: 1863 3505 1 we are called by.
: 1864 3506 1 PC_REL_CONTEXT -The number of bytes that we should encode
: 1865 3507 1 into the output byte stream to correspond to
: 1866 3508 1 a PC-relative (literal) operand.
: 1867 3509 1 BRANCH_SIZE -The number of bytes that we should allow
: 1868 3510 1 if the current operand tries to use branch
: 1869 3511 1 type addressing. 0 => do not allow branch operands.
: 1870 3512 1 OUT_PC_PTR -A pointer to the pointer that we maintain that
: 1871 3513 1 indicates where the next byte of instruction we
: 1872 3514 1 generate will go. This is used to calculate
: 1873 3515 1 PC-displacement values.
: 1874 3516 1
: 1875 3517 1 Implicit Inputs:
: 1876 3518 1
: 1877 3519 1 None.
: 1878 3520 1
: 1879 3521 1 Outputs:
: 1880 3522 1
: 1881 3523 1 The bytes that correspond to the operand reference are
: 1882 3524 1 stuffed into the vector pointed to by the pointer
: 1883 3525 1 contained in the location pointed to by OUT_BYTE_STREAM.
: 1884 3526 1 This pointer is also updated so that it points to the
: 1885 3527 1 next vacant byte position.
: 1886 3528 1
: 1887 3529 1 Routine Value:
: 1888 3530 1
: 1889 3531 1 TRUE - If successfully encoded.
: 1890 3532 1 PAT$K_BR_RANGE - If branch offset exceeds range allowed for instruction.
: 1891 3533 1 SIGNALs if an error is found and returns FALSE for the caller to
: 1892 3534 1 output the instruction.
: 1893 3535 1
: 1894 3536 1 Side Effects:
: 1895 3537 1
: 1896 3538 1 None.
: 1897 3539 1 --
: 1898 3540 1
: 1899 3541 2 BEGIN
: 1900 3542 2 MAP
: 1901 3543 2
: 1902 3544 2 ++
: 1903 3545 2 The reason why the following 2 are REFs to LONGs instead of to BYTEs
: 1904 3546 2 is because they are actually REF REF VECTOR[.BYTE], which we can only
: achieve (so far!?) via a REF LONG. Even this only works because LONG
```

```

: 1905      3547 2      ! happens to be the size of a REF BYTE (or of any REF, for that matter).
: 1906      3548 2      !--
: 1907      3549 2      OPINFO_PTR : REF BLOCK[OPTSIZE, BYTE],
: 1908      3550 2      OUT_PC_PTR : REF VECTOR[LONG],
: 1909      3551 2      OUT_BYTE_STREAM : REF VECTOR[LONG],
: 1910      3552 2      INST_STG_DESC : REF BLOCK[, BYTE];
: 1911      3553 2
: 1912      3554 2      LOCAL
: 1913      3555 2      TOKEN_STRING : VECTOR[CHS_PER_LEXEME, BYTE],
: 1914      3556 2      LEXEME_BUFFER : VECTOR[CHS_PER_LEXEME, BYTE],
: 1915      3557 2      lexeme_stg_desc : BLOCK[12, BYTE],
: 1916      3558 2      OUT_BYTE_PTR : REF VECTOR[BYTE],
: 1917      3559 2      TOKEN_TYPE : BYTE,
: 1918      3560 2      AT_FLAG,
: 1919      3561 2      MODE;
: 1920      3562 2
: 1921      3563 2      BIND
: 1922      3564 2      TOKEN_LONG = TOKEN_STRING : VECTOR[LONG];
: 1923      3565 2
: 1924      3566 2      MACRO
: 1925      3567 2      !++
: 1926      3568 2      ! This macro is used to call the routine to check for indexing and
: 1927      3569 2      ! actually output the bytes of 'instruction' into the instruction
: 1928      3570 2      ! stream. The reason why we use this macro is because it inserts the
: 1929      3571 2      ! first 2 parameters for us, (we may later use GLOBALs for this),
: 1930      3572 2      ! and because we may later have to make the control string parameter a
: 1931      3573 2      ! counted string if we find that 4 characters (a longword) are not enough.
: 1932      3574 2      !--
: 1933      3575 2      OUT_CODE ( CTRL_STRING ) =
: 1934      3576 2      BEGIN
: 1935      3577 2      IF (NOT INST_OUTPUT ( .OUT_BYTE_STREAM,
: 1936      3578 2      .OUT_PC_PTR,
: 1937      3579 2      .INST_STG_DESC,
: 1938      3580 2      CTRL_STRING,
: 1939      3581 2      %REMAINING
: 1940      3582 2      ) )
: 1941      3583 2      THEN
: 1942      3584 2      RETURN FALSE;
: 1943      3585 2      END
: 1944      3586 2      %,
: 1945      3587 2
: 1946      3588 2      !++
: 1947      3589 2      ! How we usually make up the dominant mode addressing byte.
: 1948      3590 2      !--
: 1949      3591 2      MAKE_A_MODE ( DMODE ) =
: 1950      3592 2      ( (DMODE ^ 4) OR .TOKEN_STRING[0] )
: 1951      3593 2      %,
: 1952      3594 2
: 1953      3595 2      !++
: 1954      3596 2      ! This macro creates the dominant mode for PC addressing.
: 1955      3597 2      !--
: 1956      3598 2      MAKE_PC_MODE ( DMODE ) =
: 1957      3599 2      ( (DMODE ^ 4) OR PC_REG )
: 1958      3600 2      %;
: 1959      3601 2
: 1960      3602 2      !++
: 1961      3603 2      ! Fetch the first token from the instruction string, and take action depending
```

```

1962 3604 2 | on it. If the first token is an 'at' sign, simply set a flag for later
1963 3605 2 | reference, extract the next token, and continue on.
1964 3606 2 |--
1965 3607 2 AT FLAG = FALSE;
1966 3608 2 OUT_BYTE_PTR = .OUT_BYTE_STREAM[0];
1967 3609 2 IF (TOKEN_TYPE = GET_NEXT_TOKEN(.INST_STG_DESC, TOKEN_STRING)) EQL AT_SIGN_TOKEN)
1968 3610 2 THEN
1969 3611 2 BEGIN
1970 3612 2 AT FLAG = TRUE;
1971 3613 2 TOKEN_TYPE = GET_NEXT_TOKEN(.INST_STG_DESC, TOKEN_STRING);
1972 3614 2 END;
1973 3615 2 |--
1974 3616 2 |++
1975 3617 2 | Enforce branch-type syntax only when a branch operand
1976 3618 2 | is expected, and vice versa.
1977 3619 2 |--
1978 3620 2 IF (.TOKEN_TYPE NEQ BRANCH_TOKEN)
1979 3621 2 THEN
1980 3622 2 IF (.BRANCH_SIZE NEQ 0)
1981 3623 2 THEN
1982 3624 2 BEGIN
1983 3625 2 SIGNAL(PATS_NOBRANCH+MSG$K_INFO); ! 'Branch Operand Expected' error.
1984 3626 2 RETURN(FALSE);
1985 3627 2 END;
1986 3628 2 |--
1987 3629 2 SELECTONE .TOKEN_TYPE OF
1988 3630 2 SET
1989 3631 2 [REGISTER_TOKEN]:
1990 3632 2 |--
1991 3633 2 |++
1992 3634 2 | The operand is Rn, where 'n' was returned in the TOKEN_STRING.
1993 3635 2 | Output the proper addressing mode byte, and increment the
1994 3636 2 | instruction stream pointer. Note that indexing is not allowed
1995 3637 2 | to follow REGISTER. Neither is the old MACRO11 notion of @Rx.
1996 3638 2 |--
1997 3639 2 |--
1998 3640 2 BEGIN
1999 3641 2 IF .AT_FLAG
2000 3642 2 THEN
2001 3643 2 BEGIN
2002 3644 2 SIGNAL(PATS_OPSYNTAX+MSG$K_INFO); ! '@Rx not equivalent to (Rx)' error.
2003 3645 2 RETURN(FALSE);
2004 3646 2 END
2005 3647 2 ELSE
2006 3648 2 OUT_CODE ( 'NB', (REGISTER_AMODE^4) OR .TOKEN_STRING[0] );
2007 3649 2 END;
2008 3650 2 |--
2009 3651 2 [ABS_LIT_TOKEN]:
2010 3652 2 |++
2011 3653 2 | This is either 'short literal', 'PC-Relative Literal', or
2012 3654 2 | Absolute addressing, depending on whether the string began
2013 3655 2 | with '@' or not, and on the number of bits needed to encode
2014 3656 2 | the number.
2015 3657 2 |--
2016 3658 2 BEGIN
2017 3659 2 IF .AT_FLAG
2018 3660 2 THEN
```

2019	3661	3
2020	3662	3
2021	3663	3
2022	3664	3
2023	3665	3
2024	3666	3
2025	3667	3
2026	3668	3
2027	3669	3
2028	3670	4
2029	3671	3
2030	3672	3
2031	3673	3
2032	3674	3
2033	3675	3
2034	3676	3
2035	3677	3
2036	3678	3
2037	3679	3
2038	3680	3
2039	3681	3
2040	3682	4
2041	3683	4
2042	3684	4
2043	3685	4
2044	3686	4
2045	3687	4
2046	3688	4
2047	3689	4
2048	3690	4
2049	3691	4
2050	3692	4
2051	3693	4
2052	3694	5
2053	3695	4
2054	3696	5
2055	3697	5
2056	3698	5
2057	3699	5
2058	3700	5
2059	3701	5
2060	3702	5
2061	3703	5
2062	3704	5
2063	3705	4
2064	3706	4
2065	3707	4
2066	3708	4
2067	3709	4
2068	3710	4
2069	3711	4
2070	3712	4
2071	3713	5
2072	3714	5
2073	3715	4
2074	3716	4
2075	3717	4

P
P
P

ELSE

```
++
Absolute addressing is actually a PC-relative
mode with longword context. Note that this
one may be followed by [ Rx ], which means
that we have indexing.
--
OUT_CODE( 'YBD'
MAKE_PC_MODE(AT_PC_REL_MODE),
A_LONGWORD,
TOKEN_STRING)

++
This is an immediate operand. If it will fit into 6
bits, generate 'short literal' addressing, otherwise
generate a PC-relative (immediate) mode. The check
is further qualified however, for instructions which
require greater than a byte for immediate operands in
the instruction stream.
--
IF (.TOKEN_LONG[0] GTRU 63) OR .PAT$GL_CONTEXT[I_HAT_SEEN]
THEN
BEGIN
++
The literal is too big for a 6-bit field or
a byte context is called for with I^#literal.
Therefore use a PC-relative mode and insert
the literal into the instruction stream.
Unfortunately, the number of bits to use is
NOT a function of how large the literal is,
instead this is dictated by the so-called
'context' of this instruction.
This 'context' is evidenced via the I^ operator.
--
IF (.PC_REL_CONTEXT GTR A_LONGWORD)
THEN
BEGIN
++
QUADword literals not supported.
GET_NEXT_TOKEN (i.e. PAT$RADX_CONVRT)
would have to be changed to insert 8
bytes into TOKEN_STRING.
--
SIGNAL (PAT$NOTDONE+MSG$K_INFO);
RETURN(FALSE);
END;

++
See if truncation will occur by checking
on whether the longword which the number is
taken from is different from the number as
extracted by the hardware - ie, with sign extension.
--
IF (.TOKEN_LONG[0] NEQ
(TOKEN_STRING[0])<0,.PC_REL_CONTEXT*BITS_PER_BYTE, 1>)
THEN
! The following code may be eliminated
! when RADX_CONVRT takes the length into
```

2076 3718 4
2077 3719 4
2078 3720 4
2079 3721 4
2080 3722 4
2081 3723 4
2082 3724 5
2083 3725 4
2084 3726 5
2085 3727 5
2086 3728 5
2087 3729 4
2088 3730 4
2089 3731 4
2090 3732 4
2091 3733 4
2092 3734 4
2093 3735 4
2094 3736 4
2095 3737 4
2096 3738 4
2097 3739 4
2098 3740 4
2099 3741 3
2100 3742 3
2101 3743 3
2102 3744 3
2103 3745 3
2104 3746 3
2105 3747 3
2106 3748 3
2107 3749 3
2108 3750 3
2109 3751 2
2110 3752 2
2111 3753 2
2112 3754 2
2113 3755 3
2114 3756 3
2115 3757 3
2116 3758 3
2117 3759 3
2118 3760 3
2119 3761 3
2120 3762 3
2121 3763 3
2122 3764 3
2123 3765 3
2124 3766 3
2125 3767 4
2126 3768 3
2127 3769 4
2128 3770 4
2129 3771 4
2130 3772 4
2131 3773 4
2132 3774 4

P
P
P

```
account. For now a further check is
made to see if the bits discarded
simply weren't given.
--
INCR I FROM .PC_REL_CONTEXT TO LONG_LENGTH -1
DO
  IF (.TOKEN_STRING[I] NEQ 0)
  THEN
    BEGIN
      SIGNAL (PAT$_NUMTRUNC);
      EXITLOOP;
    END;

    ++
    SRM says that #constant[Rx] is
    supported, however, to be like MARS,
    PATCH generates an error for it.
    --
    OUT_CODE( 'NBD',
              MAKE_PC_MODE( PC_REL_MODE ),
              .PC_REL_CONTEXT,
              TOKEN_STRING);
    END
  ELSE
    ++
    Short literals have to fit in 6 bits.
    They also can not be indexed.
    --
    OUT_CODE( 'NB', MAKE_A_MODE( SHORT_LIT_AMODE ) );

    ++
    Reset the context flag whether or not it has been set previously.
    --
    PAT$GL_CONTEXT[I_HAT_SEEN] = FALSE;
    END;

[BRANCH_TOKEN]:
    ! For branch type operand addressing,
    ! and for assumed PC-displacement addressing.
    BEGIN
      BIND
        ACTUAL_OPRND = TOKEN_STRING[1] : VECTOR[,LONG];

    ++
    Check the flag passed in byte 0 of the token string. A zero
    here means that the associated number is a branch operand
    exactly what is to be placed in the instruction. A 1 here
    means that the number must be made into a PC-relative offset
    - ie, it is absolute, and that it may or may not be a branch
    operand.
    --
    IF (.TOKEN_STRING[0]) AND (.OPINFO_PTR[OP_NUMOPS] NEQ ASM_DIR_OP)
    THEN
      BEGIN
        ++
        To calculate the PC-relative value, start with a
        pointer to where the actual operand will be placed,
        add in the length of the operand because that will
        be the VAX PC after it has been used to
```

```
2133 3775 4
2134 3776 4
2135 3777 4
2136 3778 4
2137 3779 4
2138 3780 4
2139 3781 5
2140 3782 4
2141 3783 4
2142 3784 4
2143 3785 4
2144 3786 4
2145 3787 4
2146 3788 4
2147 3789 4
2148 3790 4
2149 3791 4
2150 3792 4
2151 3793 5
2152 3794 4
2153 3795 3
2154 3796 3
2155 3797 3
2156 3798 3
2157 3799 3
2158 3800 3
2159 3801 4
2160 3802 3
2161 3803 4
2162 3804 4
2163 3805 4
2164 3806 4
2165 3807 4
2166 3808 4
2167 3809 4
2168 3810 4
2169 3811 4
2170 3812 4
2171 3813 4
2172 3814 4
2173 3815 4
2174 3816 5
2175 3817 4
2176 3818 5
2177 3819 6
2178 3820 6
2179 3821 5
2180 3822 5
2181 3823 5
2182 3824 5
2183 3825 6
2184 3826 6
2185 3827 6
2186 3828 6
2187 3829 5
2188 3830 5
2189 3831 4
```

```
pick up the operand, and then subtract
from that the absolute (virtual) destination.
This gives the number of bytes which have to be
added to the PC to get the address of the destination
--
ACTUAL_OPRND[0] = .ACTUAL_OPRND[0] - (.OUT_PC_PTR[0] + .BRANCH_SIZE);
IF (.BRANCH_SIZE EQL 0)
THEN
    ++
    If BRANCH_SIZE is not the right size
    (of displacement) to add, assume and use
    LONGWORD displacement. Take into consideration
    the fact that there will be a 1-byte MODE
    field before the displacement, and perhaps
    also a 1-byte index field, as well as the
    displacement itself.
    --
    ACTUAL_OPRND[0] = .ACTUAL_OPRND[0]
    - (A_LONGWORD + A_BYTE +
    ASSUME_AT_PC(.INST_STG_DESC));
END;

++
Check for branch overflow - the user trying to branch further
than the instruction can 'reach'. First, check this is a branch.
--
IF (.BRANCH_SIZE NEQ 0)
THEN
    BEGIN
    ++
    Overflow happens when all bits in the unused part of
    the LONG which we are using to contain the branch
    operand are not the same as the 'sign' bit of the
    branch operand. We find this out by extracting the
    branch operand as the hardware would (ie, with sign
    extension), and then comparing to see if this is the
    same as what we have calculated.
    For assembler directives, there is no such thing as
    the 'sign bit'. Therefore the test is made with a
    zero extend extraction.
    --
    IF (.OPINFO_PTR[OP_NUMOPS] NEQ ASM_DIR_OP)
    THEN
        BEGIN
        IF (.ACTUAL_OPRND[0] NEQ
        .(ACTUAL_OPRND)<0,.BRANCH_SIZE*BITS_PER_BYTE, 1>)
        THEN
            ++
            'Branching Out-of-Range' error.
            --
            BEGIN
            PAT$GL_BR_DISPL = .ACTUAL_OPRND[0];
            OUT_CODE('ND', .BRANCH_SIZE, ACTUAL_OPRND[0]);
            RETURN(PAT$K_BR_RANGE);
            END;
        END
    ELSE
        END
```

```

2190 3832 5
2191 3833 6
2192 3834 6
2193 3835 5
2194 3836 5
2195 3837 4
2196 3838 4
2197 3839 4
2198 3840 4
2199 3841 4
2200 3842 4
2201 3843 4
2202 3844 4
2203 3845 3
2204 3846 4
2205 3847 4
2206 3848 4
2207 3849 4
2208 3850 4
2209 3851 4
2210 3852 4
2211 3853 4
2212 3854 4
2213 3855 4
2214 3856 4
2215 3857 4
2216 3858 4
2217 3859 4
2218 3860 4
2219 3861 4
2220 3862 4
2221 3863 4
2222 3864 4
2223 3865 4
2224 3866 4
2225 3867 4
2226 3868 4
2227 3869 3
2228 3870 2
2229 3871 2
2230 3872 2
2231 3873 2
2232 3874 3
2233 3875 3
2234 3876 3
2235 3877 3
2236 3878 3
2237 3879 4
2238 3880 3
2239 3881 4
2240 3882 4
2241 3883 4
2242 3884 3
2243 3885 3
2244 3886 3
2245 3887 3
2246 3888 3

BEGIN
  IF (.ACTUAL_OPRND[0] NEQ
      .(ACTUAL_OPRND)<0,.BRANCH_SIZE*BITS_PER_BYTE, 0>)
  THEN
    SIGNAL(PAT$_NUMTRUNC);
  END;

  ++
  Branch operand is OK. Output
  the code and don't allow indexing.
  --
  OUT_CODE('ND', .BRANCH_SIZE, ACTUAL_OPRND[0]);
  END

ELSE
  BEGIN
    ++
    PC-displacement operands are similar at this point
    except that indexing must be allowed and longword
    (deferred) displacement. This code is the same as
    that at the end of case [BYTE_VAL_TOKEN], etc, below;
    PATCH relies on the compiler to combine the code rather
    than putting it into a special-purpose routine.
    --
    LEXEME_BUFFER[0] = PC_REG;
    TOKEN_STRING[0] = A_LONGWORD;
    MODE = DISP_LONG_AMODE;
    IF .AT_FLAG
    THEN
      MODE = .MODE +1;

    ++
    Pass back the single mode byte followed by the counted
    byte stream calculated. Indexing is allowed in all cases.
    --
    OUT_CODE('YBC',
      ((.MODE^4) OR .LEXEME_BUFFER[0]),
      TOKEN_STRING);
    END;
  END;

[MINUS_TOKEN]:
  BEGIN
    ++
    This must be auto decrement, '-(Rn)', or
    auto decrement indexed, '-(Rn)[Rx]'.
    --
    IF (GET_NEXT_TOKEN(.INST_STG_DESC, TOKEN_STRING) NEQ AT_REG_TOKEN)
    THEN
      BEGIN
        SIGNAL(PAT$_OPSYNTEX+MSG$_K_INFO);      ! (Rn) required for auto-decrement
        RETURN(FALSE);
      END;

    ++
    Check for indexing and output the instruction.
    --
```



```

2247 3889 3
2248 3890 3
2249 3891 4
2250 3892 4
2251 3893 4
2252 3894 4
2253 3895 4
2254 3896 4
2255 3897 4
2256 3898 3
2257 3899 3
2258 3900 2
2259 3901 2
2260 3902 2
2261 3903 2
2262 3904 3
2263 3905 3
2264 3906 3
2265 3907 3
2266 3908 3
2267 3909 3
2268 3910 3
2269 3911 3
2270 3912 3
2271 3913 3
2272 3914 3
2273 3915 3
2274 3916 3
2275 3917 3
2276 3918 3
2277 3919 3
2278 3920 3
2279 3921 3
2280 3922 4
2281 3923 3
2282 3924 4
2283 3925 4
2284 3926 4
2285 3927 4
2286 3928 4
2287 3929 4
2288 3930 4
2289 3931 4
2290 3932 4
2291 3933 4
2292 3934 4
2293 3935 4
2294 3936 4
2295 3937 3
2296 3938 3
2297 3939 3
2298 3940 4
2299 3941 4
2300 3942 4
2301 3943 3
2302 3944 3
2303 3945 3

IF .AT_FLAG
THEN
    BEGIN
        ++
        'Deferred Auto Decrement Not Allowed' error.
        --
        SIGNAL(PAT$ OPSYNTAX+MSG$K_INFO);
        RETURN(FALSE);
    END
ELSE
    OUT_CODE('YB', MAKE_A_MODE(AUTO_DEC_AMODE));
END;

[AT_REG_TOKEN]:
BEGIN
    LOCAL
        INPUT_PTR,
        CHAR;

    ++
    'This form is either register deferred, '(reg)', auto increment,
    '(reg)+', auto increment deferred, '@(reg)+', or any one of
    these plus indexing.
    --
    MODE = REG_DEF_AMODE;

    ++
    'A following '+' indicates one of the auto inc modes.
    --
    INPUT_PTR = .INST_STG_DESC [dsc$a_pointer];
    CHAR = (H$RCHAR (.INPUT_PTR));
    IF (.CHAR EQL '+')
    THEN
        BEGIN
            ++
            'Update the counted-string pointer,
            and decide which auto inc mode we have.
            --
            MODE = AUTO_INC_AMODE;
            IF .AT_FLAG
            THEN
                MODE = .MODE + 1;
                ! MODE = AUTO_INC_DEF_AMODE; ! Generates longer code.
                INST_STG_DESC [dsc$a_pointer] = CH$PLUS (.INPUT_PTR, 1);
                INST_STG_DESC [DSC$W_LENGTH] = .INST_STG_DESC [DSC$W_LENGTH] - 1;
            END
        ELSE
            IF .AT_FLAG
            THEN
                BEGIN
                    SIGNAL(PAT$ OPSYNTAX+MSG$K_INFO); ! '@(Rn) Not Supported' error.
                    RETURN(FALSE);
                END;
            ++

```

```
2304 3946 3
2305 3947 3
2306 3948 3
2307 3949 2
2308 3950 2
2309 3951 2
2310 3952 2
2311 3953 2
2312 3954 3
2313 3955 3
2314 3956 3
2315 3957 3
2316 3958 3
2317 3959 3
2318 3960 3
2319 3961 3
2320 3962 3
2321 3963 3
2322 3964 3
2323 3965 3
2324 3966 3
2325 3967 3
2326 3968 3
2327 3969 4
2328 3970 3
2329 3971 4
2330 3972 4
2331 3973 4
2332 3974 4
2333 3975 4
2334 3976 4
2335 3977 4
2336 3978 4
2337 3979 4
2338 3980 4
2339 3981 4
2340 3982 4
2341 3983 4
2342 3984 4
2343 3985 4
2344 3986 4
2345 3987 4
2346 3988 4
2347 3989 4
2348 3990 4
2349 3991 4
2350 3992 4
2351 3993 4
2352 3994 4
2353 3995 4
2354 3996 4
2355 3997 4
2356 3998 4
2357 3999 5
2358 4000 6
2359 4001 6
2360 4002 5
```

```
! In all cases PATCH allows indexing.
!--
OUT_CODE( 'YB', MAKE_A_MODE( .MODE ) );
END;

[BYTE_VAL_TOKEN,
WORD_VAL_TOKEN,
LONG_VAL_TOKEN]:
BEGIN
  BIND
    ACTUAL_OPRND = TOKEN_STRING[1] : VECTOR[,LONG];
  LOCAL
    INDEXING;

  ++
  Displacement or Deferred Displacement addressing.

  Here PATCH must 'look ahead' to see if an actual register to
  displace off has been given. If not, assume "(PC)" and
  treat the displacement as a virtual address, calculating what
  real displacement must be used given that the PC is the same
  as the address into which the instruction is being deposited.
  --
  IF ((INDEXING = ASSUME_AT_PC(.INST_STG_DESC)) GEQ 0)
  THEN
    BEGIN
      ++
      Ok to assume PC-displacement mode. This means that
      the given displacement is the virtual address to be
      reached, so this field must be converted to a real
      displacement. To do this PATCH calculates what the PC
      will be after it has been used to pick up the
      displacement to find out how much this displacement
      must be. The INDEXING value returned above indicates
      how many bytes will be output due to indexed addressing.
      --
      ACTUAL_OPRND[0] = .ACTUAL_OPRND[0]
        - (.OUT_PC_PTR[0] + .TOKEN_STRING[0] + A_BYTE + .INDEXING);

      ++
      Also fake the user having said "(PC)"
      by filling in LEXEME_BUFFER with the
      right AT_REGISTER name.
      --
      LEXEME_BUFFER[0] = PC_REG;

      ++
      Check for trying to branch too far. Here, the check
      must be ensuring that the unused bits in the LONG
      version of ACTUAL_OPRND are the same as the sign bit
      of the actual displacement part of ACTUAL_OPRND.
      --
      REPEAT
        BEGIN
          IF (.ACTUAL_OPRND[0] NEQ
            .(ACTUAL_OPRND)<0,.TOKEN_STRING[0]*BITS_PER_BYTE, 1>)
          THEN
```

```
: 2361 4003 5
: 2362 4004 6
: 2363 4005 5
: 2364 4006 6
: 2365 4007 6
: 2366 4008 7
: 2367 4009 6
: 2368 4010 7
: 2369 4011 7
: 2370 4012 7
: 2371 4013 7
: 2372 4014 6
: 2373 4015 7
: 2374 4016 7
: 2375 4017 7
: 2376 4018 6
: 2377 4019 6
: 2378 4020 6
: 2379 4021 5
: 2380 4022 6
: 2381 4023 6
: 2382 4024 6
: 2383 4025 6
: 2384 4026 5
: 2385 4027 5
: 2386 4028 4
: 2387 4029 4
: 2388 4030 3
: 2389 4031 4
: 2390 4032 4
: 2391 4033 4
: 2392 4034 4
: 2393 4035 4
: 2394 4036 5
: 2395 4037 4
: 2396 4038 5
: 2397 4039 5
: 2398 4040 5
: 2399 4041 4
: 2400 4042 4
: 2401 4043 4
: 2402 4044 4
: 2403 4045 4
: 2404 4046 4
: 2405 4047 4
: 2406 4048 4
: 2407 4049 4
: 2408 4050 4
: 2409 4051 4
: 2410 4052 4
: 2411 4053 4
: 2412 4054 4
: 2413 4055 5
: 2414 4056 5
: 2415 4057 4
: 2416 4058 4
: 2417 4059 4
```

```
IF (.PAT$GL_CONTEXT[INST_SUBST]) AND
  (.TOKEN_STRING[0] LSS LONG_LENGTH)
THEN
  BEGIN
    ACTUAL_OPRND[0] = .ACTUAL_OPRND[0] + .TOKEN_STRING[0];
    IF (.TOKEN_TYPE EQL BYTE_VAL_TOKEN)
    THEN
      BEGIN
        TOKEN_STRING[0] = WORD_LENGTH;
        TOKEN_TYPE = WORD_VAL_TOKEN;
      END
    ELSE
      BEGIN
        TOKEN_STRING[0] = LONG_LENGTH;
        TOKEN_TYPE = LONG_VAL_TOKEN;
      END;
    ACTUAL_OPRND[0] = .ACTUAL_OPRND[0] - .TOKEN_STRING[0];
  END
ELSE
  BEGIN
    SIGNAL(PAT$_BRTOOFAR+MSG$K_INFO, 1, .ACTUAL_OPRND[0]);
    EXITLOOP;
  END
ELSE
  END
  BEGIN
    ++
    ! Check that the displacement is followed
    ! by a register reference in parenthesis.
    --
    IF (GET_NEXT_TOKEN(.INST_STG_DESC, LEXEME_BUFFER) NEQ AT_REG_TOKEN)
    THEN
      BEGIN
        SIGNAL(PAT$ OPSYNTAX+MSG$K_INFO); ! 'Must Displace off a Reg' error.
        RETURN(FALSE);
      END;

      ++
      ! Check for displacement truncation and produce a
      ! message if this will occur. Here the check is based
      ! on the sign bit of the actual displacement, as done
      ! to check this above. Here, however, check whether or
      ! not the upper bits of the given displacement are all
      ! 0, and 'forgive' if this is true. This nonsense is
      ! necessary to avoid complaining when one says
      ! 'B*95(reg)'. Here the 95 is taken as a negative
      ! number rather than assuming a large positive one was
      ! intended.
      --
      IF (.ACTUAL_OPRND[0] NEQ
        (.ACTUAL_OPRND[0] < 0, .TOKEN_STRING[0]*BITS_PER_BYTE, 1 >))
      THEN
        ++
        ! When RADX_CONVRT is changed to take size into
```

```
2418 4060 4
2419 4061 4
2420 4062 4
2421 4063 4
2422 4064 4
2423 4065 5
2424 4066 5
2425 4067 5
2426 4068 5
2427 4069 5
2428 4070 5
2429 4071 5
2430 4072 6
2431 4073 5
2432 4074 6
2433 4075 6
2434 4076 7
2435 4077 8
2436 4078 8
2437 4079 7
2438 4080 7
2439 4081 8
2440 4082 7
2441 4083 8
2442 4084 8
2443 4085 9
2444 4086 8
2445 4087 9
2446 4088 9
2447 4089 9
2448 4090 9
2449 4091 8
2450 4092 9
2451 4093 9
2452 4094 9
2453 4095 8
2454 4096 8
2455 4097 8
2456 4098 7
2457 4099 8
2458 4100 8
2459 4101 8
2460 4102 8
2461 4103 7
2462 4104 7
2463 4105 6
2464 4106 6
2465 4107 5
2466 4108 4
2467 4109 3
2468 4110 3
2469 4111 3
2470 4112 3
2471 4113 3
2472 4114 3
2473 4115 3
2474 4116 3
```

```
! account (ie, to complain if the number is too
! big), just produce a message at this point.
! For now, however, ignore it if the unused
! bytes are all zero.
```

```
BEGIN
```

```
BIND
```

```
ACTUAL_BYTES = ACTUAL_OPRND[0] : VECTOR[,BYTE];
```

```
INCR I FROM .TOKEN_STRING[0] TO LONG_LENGTH -1
```

```
DO
```

```
IF(.ACTUAL_BYTES[.I] NEQ 0)
```

```
THEN
```

```
BEGIN
REPEAT
```

```
BEGIN
```

```
IF (.ACTUAL_OPRND[0] NEQ
```

```
.(ACTUAL_OPRND)<0,.TOKEN_STRING[0]*BITS_PER_BYTE
```

```
THEN
```

```
IF (.PAT$GL_CONTEXT[INST_SUBST]) AND
(.TOKEN_STRING[0] LSS LONG_LENGTH)
THEN
```

```
BEGIN
```

```
ACTUAL_OPRND[0] = .ACTUAL_OPRND[0] +
IF (.TOKEN_TYPE EQL BYTE_VAL_TOKEN)
THEN
```

```
BEGIN
```

```
TOKEN_STRING[0] = WORD_LENGTH
TOKEN_TYPE = WORD_VAL_TOKEN;
END
```

```
ELSE
```

```
BEGIN
```

```
TOKEN_STRING[0] = LONG_LENGTH
TOKEN_TYPE = LONG_VAL_TOKEN;
END;
```

```
ACTUAL_OPRND[0] = .ACTUAL_OPRND[0] -
END
```

```
ELSE
```

```
BEGIN
```

```
SIGNAL(PAT$_BRTOOFAR+MSG$K_INFO, 1,
EXITLOOP;
END
```

```
ELSE
```

```
EXITLOOP;
```

```
END;
```

```
EXITLOOP;
END;
```

```
END;
```

```
END;
```

```
!++
```

```
Now calculate the right mode to use. The following code is
extremely instruction-set dependent, and relies on the
relative values of the various displacement modes.
Essentially just start out with the lowest mode and keep
incrementing the mode. This code is shorter but relies
on the relationship of the modes.
```

```

: 2475      4117 3      !--
: 2476      4118 3      MODE = DISP BYTE AMODE;
: 2477      4119 4      IF (.TOKEN_TYPE GTR BYTE_VAL_TOKEN)
: 2478      4120 3      THEN
: 2479      4121 3          MODE = .MODE +2;
: 2480      4122 4      IF (.TOKEN_TYPE GTR WORD_VAL_TOKEN)
: 2481      4123 3      THEN
: 2482      4124 3          MODE = .MODE +2;
: 2483      4125 3      IF .AT_FLAG
: 2484      4126 3      THEN
: 2485      4127 3          MODE = .MODE +1;
: 2486      4128 3
: 2487      4129 3      !++
: 2488      4130 3      ! Return the single mode byte followed by the counted byte
: 2489      4131 3      ! stream passed. Indexing is allowed in all cases.
: 2490      4132 3      !--
: 2491      4133 3      OUT_CODE( 'YBC',
P      4134 3          ((.MODE^4) OR .LEXEME_BUFFER[0]),
P      4135 3          TOKEN_STRING);
: 2492      4136 3      END;
: 2493      4137 2
: 2494      4138 2      [OTHERWISE]:
: 2495      4139 2          ! Error.
: 2496      4140 3
: 2497      4141 3      BEGIN
: 2498      4142 3      SIGNAL(PAT$ OPSYNTAX+MSG$K_INFO);
: 2499      4143 3          ! 'Operand Syntax' error.
: 2500      4144 3      RETURN(FALSE);
: 2501      4145 2      END;
: 2502      4146 2
: 2503      4147 2      TES;
: 2504      4148 2
: 2505      4149 2      !++
: 2506      4150 2      ! Return a pointer to the counted string which contains the rest of the operand
: 2507      4151 2      ! reference.
: 2508      4152 2      !--
: 2509      4153 2      RETURN TRUE
: 2510      4154 1      END;
```

```

OFFC 00000 ENC_OPERAND:
5B      F601      CF      9E      00002      .WORD      Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      : 3490
5A      0000C_00G      00      9E      00007      MOVAB      GET NEXT TOKEN, R11      :
59      00000000G      EF      9E      0000E      MOVAB      LIB$SIGNAL, R10      :
58      00000000V      EF      9E      00015      MOVAB      PAT$GL_CONTEXT, R9      :
5E      BC      AE      9E      0001C      MOVAB      INST_OUTPUT, R8      :
57      D4      00020      CLRL      AT_FLAG      : 3607
56      08      AC      D0      00022      MOVL      OUT_BYTE_STREAM, R6      : 3608
50      66      D0      00026      MOVL      (R6), OUT_BYTE_PTR      :
53      28      AE      9F      00029      PUSHAB      TOKEN_STRING      : 3609
54      04      AC      D0      0002C      MOVL      INST_STG_DESC, R3      :
55      02      FB      00032      PUSHL      R3      :
56      50      90      00035      CALLS      #2, GET NEXT TOKEN      :
57      D1      00038      MOVAB      R0, TOKEN_TYPE      :
58      CMPL      R0, #61      :
```

			0E	12	0003B	BNEQ	1\$	:		
	57		01	DD	0003D	MOVL	#1, AT_FLAG	:	3612	
		28	AE	9F	00040	PUSHAB	TOKEN_STRING	:	3613	
			53	DD	00043	PUSHL	R3	:		
	6B		02	FB	00045	CALLS	#2, GET_NEXT_TOKEN	:		
	55		50	90	00048	MOVB	R0, TOKEN_TYPE	:		
F4	8F		55	91	0004B	1\$:	CMPB	TOKEN_TYPE, #244	3620	
			DD	13	0004F		BEQL	2\$		
		10	AC	D5	00051		TSTL	BRANCH_SIZE	3622	
			08	13	00054		BEQL	2\$		
		006D822B	8F	DD	00056		PUSHL	#7176747	3625	
			5E	11	0005C		BRB	9\$		
F7	8F		55	91	0005E	2\$:	CMPB	TOKEN_TYPE, #247	3632	
			24	12	00062		BNEQ	5\$		
	03		57	E9	00064		BLBC	AT_FLAG, 3\$	3641	
			033F	31	00067		BRW	58\$		
	50	28	AE	9A	0006A	3\$:	MOVZBL	TOKEN_STRING, R0	3648	
7E	50	00000050	8F	C9	0006E		BISL3	#80, R0, -(SP)		
	7E	424E	8F	3C	00076		MOVZWL	#16974, -(SP)		
			53	DD	0007B	4\$:	PUSHL	R3		
		14	AC	DD	0007D		PUSHL	OUT_PC_PTR		
			56	DD	00080		PUSHL	R6		
	6B		05	FB	00082		CALLS	#5, INST_OUTPUT		
			031C	31	00085		BRW	57\$		
F8	8F		55	91	00088	5\$:	CMPB	TOKEN_TYPE, #248	3651	
			03	13	0008C		BEQL	6\$		
			0097	31	0008E		BRW	18\$		
	11		57	E9	00091	6\$:	BLBC	AT_FLAG, 7\$	3659	
		28	AE	9F	00094		PUSHAB	TOKEN_STRING	3670	
			04	DD	00097		PUSHL	#4		
	7E		9F	8F	9A	00099	MOVZBL	#159, -(SP)		
		00444259	8F	DD	0009D		PUSHL	#4473433		
			57	11	000A3		BRB	14\$		
	3F	28	AE	D1	000A5	7\$:	CMPL	TOKEN_LONG, #63	3680	
			05	1A	000A9		BGTRU	8\$		
58	03	A9	02	E1	000AB		BBC	#2, PAT\$GL_CONTEXT+3, 15\$		
		04	AC	D1	000B0	8\$:	CMPL	PC_REL_CONTEXT, #4	3694	
			09	15	000B4		BLEQ	10\$		
		006D823B	8F	DD	000B6		PUSHL	#7176763	3703	
			02F0	31	000BC	9\$:	BRW	59\$		
			03	78	000BF	10\$:	ASHL	#3, PC_REL_CONTEXT, R1	3714	
50	28	AE	00	EE	000C4		EXTV	#0, R1, TOKEN_STRING, R0		
			AE	D1	000CA		CMPL	TOKEN_LONG, R0		
			1C	13	000CE		BEQL	13\$		
	52	OC	01	C3	000D0		SUBL3	#1, PC_REL_CONTEXT, I	3722	
			11	11	000D5		BRB	12\$		
		28	AE	42	95	000D7	11\$:	TSTB	TOKEN_STRING[I]	3724
			0B	13	000DB		BEQL	12\$		
		006D8023	8F	DD	000DD		PUSHL	#7176227	3727	
	6A		01	FB	000E3		CALLS	#1, LIB\$SIGNAL		
			C4	11	000E6		BRB	13\$		
			03	F3	000E8	12\$:	AOBLEQ	#3, I, 11\$	3726	
EB	52		AE	9F	000EC	13\$:	PUSHAB	TOKEN_STRING	3724	
		28	AC	DD	000EF		PUSHL	PC_REC_CONTEXT	3739	
		OC	8F	9A	000F2		MOVZBL	#173, -(SP)		
			8F	DD	000F6		PUSHL	#4473422		
	7E	0044424E	53	DD	000FC	14\$:	PUSHL	R3		

				14	AC	DD	000FE	PUSHL	OUT_PC_PTR			
				56	DD	00101		PUSHL	R6			
		68		07	FB	00103		CALLS	#7, INST_OUTPUT			
				13	11	00106		BRB	16\$			
		7E		28	AE	9A	00108	15\$:	MOVZBL	TOKEN_STRING, -(SP)	3746	
		7E		424E	8F	3C	0010C		MOVZWL	#16974, -(SP)		
				53	DD	00111		PUSHL	R3			
				14	AC	DD	00113		PUSHL	OUT_PC_PTR		
				56	DD	00116		PUSHL	R6			
		68		05	FB	00118		CALLS	#5, INST_OUTPUT			
		03		50	E8	0011B		16\$:	BLBS	R0, 17\$		
				0297	31	0011E			BRW	61\$		
		03	A9	04	8A	00121		17\$:	BICB2	#4, PAT\$GL_CONTEXT+3	3750	
				028C	31	00125			BRW	60\$	3629	
		F4	8F	55	91	00128		18\$:	CMPB	TOKEN_TYPE, #244	3753	
				03	13	0012C			BEQL	19\$		
				00C6	31	0012E			BRW	26\$		
		32		28	AE	E9	00131	19\$:	BLBC	TOKEN_STRING, 20\$	3767	
FFFFFFFE	8F	04	A0	50	18	AC	D0	00135	MOVL	OPINFO_PTR, R0		
				04	00	EC	00139		CMPV	#0, #4, 4(R0), #-2		
				50	22	13	00143		BEQL	20\$		
				14	AC	C1	00145		ADDL3	BRANCH_SIZE, @OUT_PC_PTR, R0	3780	
				29	50	C2	0014B		SUBL2	R0, ACTUAL_OPRND		
					10	AC	D5	0014F	TSTL	BRANCH_SIZE	3781	
					13	12	00152		BNEQ	20\$		
					53	DD	00154		PUSHL	R3	3794	
					01	FB	00156		CALLS	#1, ASSUME_AT_PC		
				50	C3	0015D			SUBL3	R0, ACTUAL_OPRND, R0	3793	
				29	FB	A0	9E	00162	MOVAB	-5(R0), ACTUAL_OPRND		
				52	10	AC	D0	00167	20\$:	MOVL	BRANCH_SIZE, R2	3801
					68	13	00168		BEQL	24\$		
				51	03	78	0016D		ASHL	#3, R2, R1	3820	
				50	18	AC	D0	00171	MOVL	OPINFO_PTR, R0	3816	
FFFFFFFE	8F	04	A0	04	00	EC	00175		CMPV	#0, #4, 4(R0), #-2		
					32	13	0017F		BEQL	22\$		
				51	00	EE	00181		EXTV	#0, R1, ACTUAL_OPRND, R0	3820	
				50	29	AE	D1	00187	CMPL	ACTUAL_OPRND, R0		
					3B	13	0018B		BEQL	23\$		
				00000000G	29	AE	D0	0018D	MOVL	ACTUAL_OPRND, PAT\$GL_BR_DISPL	3826	
					29	AE	9F	00195	PUSHAB	ACTUAL_OPRND	3827	
					52	DD	00198		PUSHL	R2		
				7E	444E	8F	3C	0019A	MOVZWL	#17486, -(SP)		
					53	DD	0019F		PUSHL	R3		
				14	AC	DD	001A1		PUSHL	OUT_PC_PTR		
					56	DD	001A4		PUSHL	R6		
				68	06	FB	001A6		CALLS	#6, INST_OUTPUT		
				03	50	E8	001A9		BLBS	R0, 21\$		
					0209	31	001AC		BRW	61\$		
				50	02	D0	001AF	21\$:	MOVL	#2, R0	3828	
					04	001B2			RET			
				51	00	EF	001B3	22\$:	EXTZV	#0, R1, ACTUAL_OPRND, R0	3834	
				50	29	AE	D1	001B9	CMPL	ACTUAL_OPRND, R0		
					09	13	001BD		BEQL	23\$		
				006D8023	8F	DD	001BF		PUSHL	#7176227	3836	
				6A	01	FB	001C5		CALLS	#1, LIB\$SIGNAL		
					29	AE	9F	001C8	23\$:	PUSHAB	ACTUAL_OPRND	3843
					52	DD	001CB		PUSHL	R2		

	7E	444E	8F	3C	001CD	MOVZWL	#17486, -(SP)	:	
			01C5	31	001D2	BRW	56\$	:	
	0C	AE	0F	90	001D5	24\$:	MOV8	#15, LEXEME_BUFFER	3855
	28	AE	04	90	001D9	MOV8	#4, TOKEN_STRING	:	3856
		52	0E	D0	001DD	MOVL	#14, MODE	:	3857
		02	57	E9	001E0	BLBC	AT FLAG, 25\$	:	3858
			52	D6	001E3	INCL	MODE	:	3860
		28	AE	9F	001E5	25\$:	PUSHAB	TOKEN_STRING	3868
50		52	04	78	001E8	ASHL	#4, MODE, R0	:	
		51	10	AE	9A	001EC	MOVZBL	LEXEME_BUFFER, R1	
7E		50	51	C9	001F0	BISL3	R1, R0, -(SP)	:	
			019D	31	001F4	BRW	55\$	:	
	46	8F	55	91	001F7	26\$:	CMPB	TOKEN_TYPE, #70	3873
			22	12	001FB	BNEQ	27\$	:	
		28	AE	9F	001FD	PUSHAB	TOKEN_STRING	:	3879
			53	DD	00200	PUSHL	R3	:	
		6B	02	FB	00202	CALLS	#2, GET_NEXT_TOKEN	:	
	000000F6	8F	50	D1	00205	CMPL	R0, #248	:	
			61	12	0020C	BNEQ	35\$	:	
		34	57	E8	0020E	BLBS	AT FLAG, 29\$	:	3889
		50	28	AE	9A	00211	MOVZBL	TOKEN_STRING, R0	3899
7E		50	00000070	8F	C9	00215	BISL3	#112, R0, -(SP)	:
				38	11	0021D	BRB	32\$	:
	F6	8F	55	91	0021F	27\$:	CMPB	TOKEN_TYPE, #246	3903
			3A	12	00223	BNEQ	33\$	:	
		52	06	D0	00225	MOVL	#6, MODE	:	3915
		50	04	A3	D0	00228	MOVL	4(R3), INPUT_PTR	3920
		51	60	9A	0022C	MOVZBL	(INPUT_PTR), CHAR	:	3921
		2B	51	D1	0022F	CMPL	CHAR, #43	:	3922
			11	12	00232	BNEQ	29\$	:	
		52	08	D0	00234	MOVL	#8, MODE	:	3929
		02	57	E9	00237	BLBC	AT FLAG, 28\$	:	3930
			52	D6	0023A	INCL	MODE	:	3932
	04	A3	01	A0	9E	0023C	28\$:	MOVAB	1(R0), 4(R3)
			63	B7	00241	DECW	(R3)	:	3934
			06	11	00243	BRB	31\$	:	3935
		03	57	E9	00245	29\$:	BLBC	AT FLAG, 31\$	3938
			015E	31	00248	30\$:	BRW	58\$	
50		52	04	78	0024B	31\$:	ASHL	#4, MODE, R0	3948
		51	28	AE	9A	0024F	MOVZBL	TOKEN_STRING, R1	:
7E		50	51	C9	00253	BISL3	R1, R0, -(SP)	:	
		7E	4259	8F	3C	00257	32\$:	MOVZWL	#16985, -(SP)
			FE1C	31	0025C	BRW	4\$	:	
	F2	8F	55	91	0025F	33\$:	CMPB	TOKEN_TYPE, #242	3951
			06	1F	00263	BLSSU	34\$	:	
	F3	8F	55	91	00265	CMPB	TOKEN_TYPE, #243	:	
			06	1B	00269	BLEQU	36\$	:	
	F5	8F	55	91	0026B	34\$:	CMPB	TOKEN_TYPE, #245	
			D7	12	0026F	35\$:	BNEQ	30\$	
			53	DD	00271	36\$:	PUSHL	R3	3969
	00000000V	EF	01	FB	00273	CALLS	#1, ASSUME_AT_PC	:	
			50	D5	0027A	TSTL	INDEXING	:	
			5F	19	0027C	BLSS	40\$	:	
		51	28	AE	9A	0027E	MOVZBL	TOKEN_STRING, R1	3983
		51	14	BC	C0	00282	ADDL2	@OUT_PC_PTR, R1	:
		50	51	C0	00286	ADDL2	R1, R0	:	
50	29	AE	50	C3	00289	SUBL3	R0, ACTUAL_OPRND, R0	:	

50	29	AE	29	AE	FF	A0	9E	0028E	MOVAB	-1(R0), ACTUAL_OPRND	3990	
			0C	AE		0F	90	00293	MOVB	#15, LEXEME_BUFFER	4001	
				51	28	AE	9A	00297	MOVZBL	TOKEN_STRING, R1		
				51		08	C4	0029B	MULL2	#8, RT		
				51		00	EE	0029E	EXTV	#0, R1, ACTUAL_OPRND, R0		
				50	29	AE	D1	002A4	CMPL	ACTUAL_OPRND, R0		
						79	13	002A8	BEQL	44\$		
		76	02	A9		04	E1	002AA	BBC	#4, PAT\$GL_CONTEXT+2, 45\$	4003	
				04	28	AE	91	002AF	CMPB	TOKEN_STRING, #4	4004	
						79	1E	002B3	BGEQU	46\$		
				50	28	AE	9A	002B5	MOVZBL	TOKEN_STRING, R0	4007	
			29	AE		50	C0	002B9	ADDL2	R0, ACTUAL_OPRND		
			F2	8F		55	91	002BD	CMPB	TOKEN_TYPE, #242	4008	
						09	12	002C1	BNEQ	38\$		
			28	AE		02	90	002C3	MOVB	#2, TOKEN_STRING	4011	
				55		0D	8E	002C7	MNEGB	#13, TOKEN_TYPE	4012	
						07	11	002CA	BRB	39\$	4008	
			28	AE		04	90	002CC	MOVB	#4, TOKEN_STRING	4016	
				55		0B	8E	002D0	MNEGB	#11, TOKEN_TYPE	4017	
				50	28	AE	9A	002D3	MOVZBL	TOKEN_STRING, R0	4019	
			29	AE		50	C2	002D7	SUBL2	R0, ACTUAL_OPRND		
						BA	11	002DB	BRB	37\$	4003	
					0C	AE	9F	002DD	PUSHAB	LEXEME_BUFFER	4036	
						53	DD	002E0	PUSHL	R3		
				6B		02	FB	002E2	CALLS	#2, GET_NEXT_TOKEN		
		000000F6		8F		50	D1	002E5	CMPL	R0, #246		
						03	13	002EC	BEQL	41\$		
					00B8	31	002EE	BRW	58\$			
				51	28	AE	9A	002F1	MOVZBL	TOKEN_STRING, R1	4056	
				51		08	C4	002F5	MULL2	#8, RT		
				51		00	EE	002F8	EXTV	#0, R1, ACTUAL_OPRND, R0		
				50	29	AE	D1	002FE	CMPL	ACTUAL_OPRND, R0		
						68	13	00302	BEQL	51\$		
				54	28	AE	9A	00304	MOVZBL	TOKEN_STRING, I	4070	
						54	D7	00308	DECL	I		
						5C	11	0030A	BRB	50\$		
					29	AE	44	95	0030C	TSTB	ACTUAL_BYTES[I]	4072
						56	13	00310	BEQL	50\$		
				51	28	AE	9A	00312	MOVZBL	TOKEN_STRING, R1	4078	
				51		08	C4	00316	MULL2	#8, RT		
				51		00	EE	00319	EXTV	#0, R1, ACTUAL_OPRND, R0		
				50	29	AE	D1	0031F	CMPL	ACTUAL_OPRND, R0		
						47	13	00323	BEQL	51\$		
						04	E1	00325	BBC	#4, PAT\$GL_CONTEXT+2, 49\$	4080	
			2E	02	28	AE	91	0032A	CMPB	TOKEN_STRING, #4	4081	
				04		28	1E	0032E	BGEQU	49\$		
				50	28	AE	9A	00330	MOVZBL	TOKEN_STRING, R0	4084	
			29	AE		50	C0	00334	ADDL2	R0, ACTUAL_OPRND		
			F2	8F		55	91	00338	CMPB	TOKEN_TYPE, #242	4085	
						09	12	0033C	BNEQ	47\$		
			28	AE		02	90	0033E	MOVB	#2, TOKEN_STRING	4088	
				55		0D	8E	00342	MNEGB	#13, TOKEN_TYPE	4089	
						07	11	00345	BRB	48\$	4085	
			28	AE		04	90	00347	MOVB	#4, TOKEN_STRING	4093	
				55		0B	8E	0034B	MNEGB	#11, TOKEN_TYPE	4094	
				50	28	AE	9A	0034E	MOVZBL	TOKEN_STRING, R0	4096	
			29	AE		50	C2	00352	SUBL2	R0, ACTUAL_OPRND		

			BA	11	00356		BRB	43\$		4080
		29	AE	DD	00358	49\$:	PUSHL	ACTUAL_OPRND		4100
			01	DD	00358		PUSHL	#1		
		006D8223	8F	DD	0035D		PUSHL	#7176739		
	6A		03	FB	00363		CALLS	#3, LIB\$SIGNAL		
			04	11	00366		BRB	51\$		4099
A0	54		03	F3	00368	50\$:	AOBLEQ	#3, I, 42\$		4072
	52		0A	D0	0036C	51\$:	MOVL	#10, MODE		4118
	F2	8F	55	91	0036F		CMPB	TOKEN_TYPE, #242		4119
			03	1B	00373		BLEQU	52\$		
	52		02	C0	00375		ADDL2	#2, MODE		4121
	F3	8F	55	91	00378	52\$:	CMPB	TOKEN_TYPE, #243		4122
			03	1B	0037C		BLEQU	53\$		
	52		02	C0	0037E		ADDL2	#2, MODE		4124
	02		57	E9	00381	53\$:	BLBC	AT_FLAG, 54\$		4125
			52	D6	00384		INCL	MODE		4127
		28	AE	9F	00386	54\$:	PUSHAB	TOKEN_STRING		4135
	52		10	C4	00389		MULL2	#16, R2		
	50	10	AE	9A	0038C		MOVZBL	LEXEME_BUFFER, R0		
7E	52		50	C9	00390		BISL3	R0, R2, -(SP)		
		00434259	8F	DD	00394	55\$:	PUSHL	#4407897		
			53	DD	0039A	56\$:	PUSHL	R3		
		14	AC	DD	0039C		PUSHL	OUT_PC_PTR		
			56	DD	0039F		PUSHL	R6		
	68		06	FB	003A1		CALLS	#6, INST_OUTPUT		
	0D		50	E8	003A4	57\$:	BLBS	R0, 60\$		
			0F	11	003A7		BRB	61\$		
		006D821B	8F	DD	003A9	58\$:	PUSHL	#7176731		4141
	6A		01	FB	003AF	59\$:	CALLS	#1, LIB\$SIGNAL		
			04	11	003B2		BRB	61\$		4142
	50		01	D0	003B4	60\$:	MOVL	#1, R0		4151
				04	003B7		RET			
			50	D4	003B8	61\$:	CLRL	R0		4152
			04	003BA			RET			

; Routine Size: 955 bytes, Routine Base: _PAT\$CODE + 0C85

```

: 2512 4153 1 ROUTINE ASSUME_AT_PC( INST_STG_DESC ) =
: 2513 4154 1
: 2514 4155 1 |++
: 2515 4156 1 | Functional Description:
: 2516 4157 1
: 2517 4158 1 | This temporary routine is called when PATCH has read the <size> ^ <number>
: 2518 4159 1 | part of an <operand reference> to decide whether the person has left out the
: 2519 4160 1 | "(pc)". This decision is made by looking ahead way to see if what follows
: 2520 4161 1 | could be legal assuming that he meant "(PC)". In this case the only
: 2521 4162 1 | possibilities are that he wants indexing, or that this is the end of
: 2522 4163 1 | the operand reference.
: 2523 4164 1
: 2524 4165 1 | Inputs:
: 2525 4166 1
: 2526 4167 1 | INST_STG_DESC - String descriptor to what is left
: 2527 4168 1 | of the operand reference after the displacement
: 2528 4169 1 | or virtual address has been extracted.
: 2529 4170 1
: 2530 4171 1 | Returned Value:
: 2531 4172 1
: 2532 4173 1 | 0 or 1, if "(PC)" can be assumed,
: 2533 4174 1 | -1, otherwise. (DO_NOT_ASSUME)
: 2534 4175 1
: 2535 4176 1 | 0 => no indexing byte will be output,
: 2536 4177 1 | 1 => one byte will be output for [ Rx ].
: 2537 4178 1
: 2538 4179 1 | In no case is any part of the instruction string changed.
: 2539 4180 1 | This routine has no side effects whatsoever.
: 2540 4181 1 |--
: 2541 4182 1
: 2542 4183 2 BEGIN
: 2543 4184 2
: 2544 4185 2 MAP
: 2545 4186 2 INST_STG_DESC : REF BLOCK [, BYTE];
: 2546 4187 2
: 2547 4188 2 LOCAL
: 2548 4189 2 INPUT_PTR : REF VECTOR[,BYTE];
: 2549 4190 2
: 2550 4191 2 MACRO
: 2551 4192 2 DO_NOT_ASSUME = -1 %; ! Value returned if unable to make PC assumpt
: 2552 4193 2
: 2553 4194 2 |++
: 2554 4195 2 | PATCH must believe the count part of the counted string and not overrun it.
: 2555 4196 2 | This should not be a problem, though, because the string ends with a null
: 2556 4197 2 | byte. Also detect whether the reason for returning OK-to-assume-"(PC)" is
: 2557 4198 2 | because the operand is going to include indexing or not. Do this based on the
: 2558 4199 2 | presence or absence of '['.
: 2559 4200 2 |--
: 2560 4201 3 IF (INST_STG_DESC[DSC$W_LENGTH] EQL 0)
: 2561 4202 2 THEN
: 2562 4203 2 RETURN(FALSE);
: 2563 4204 2 INPUT_PTR = CH$PTR (INST_STG_DESC [DSC$A_POINTER]);
: 2564 4205 3 IF (INPUT_PTR[0] EQL ASC$_SQ_OPN_BRK)
: 2565 4206 2 THEN
: 2566 4207 3 RETURN(TRUE)
: 2567 4208 2 ELSE
: 2568 4209 2 RETURN(DO_NOT_ASSUME);
```

PATENC
V04-000

; 2569

4210 1 END;

I 16
16-Sep-1984 00:40:15
14-Sep-1984 12:52:32

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[PATCH.SRC]PATENC.B32;1 Page 74
(9)

			0000	00000	ASSUME_AT	PC:		
						WORD	Save nothing	: 4153
	50	04	AC	D0	00002	MOVL	INST_STG_DESC, R0	: 4201
			60	B5	00006	TSTW	(R0)	:
			12	13	00008	BEQL	2\$	:
	50	04	A0	D0	0000A	MOVL	4(R0), INPUT_PTR	: 4204
5B	8F		60	91	0000E	CMPB	(INPUT_PTR), #91	: 4205
			04	12	00012	BNEQ	1\$	:
	50		01	D0	00014	MOVL	#1, R0	: 4209
				04	00017	RET		:
	50		01	CE	00018	MNEGL	#1, R0	:
				04	0001B	RET		:
			50	B4	0001C	CLRL	R0	: 4210
				04	0001E	RET		:

; Routine Size: 31 bytes, Routine Base: _PAT\$CODE + 1040

```
2571 4211 1 ROUTINE INST_OUTPUT ( OUT_BYTE_STREAM,  
2572 4212 1 OUT_PC_PTR,  
2573 4213 1 INST_STG_DESC,  
2574 4214 1 CTRL_STRING,  
2575 4215 1 ARG0,  
2576 4216 1 ARG1,  
2577 4217 1 ARG2  
2578 4218 1 ) =  
2579 4219 1 ++  
2580 4220 1 Functional Description:  
2581 4221 1  
2582 4222 1 This routine serves two purposes.  
2583 4223 1 1) It checks whether an indexed operand reference  
2584 4224 1 has been made, and outputs the proper mode byte  
2585 4225 1 if it has.  
2586 4226 1 2) It takes care of all other instruction byte  
2587 4227 1 output as well - not that this routine computes  
2588 4228 1 any of this - it just localizes such output.  
2589 4229 1  
2590 4230 1 Calling Sequence:  
2591 4231 1 INST_OUTPUT (  
2592 4232 1  
2593 4233 1  
2594 4234 1 Inputs:  
2595 4235 1  
2596 4236 1 OUT_BYTE_STREAM -The address of a pointer to where  
2597 4237 1 we are in the output stream. The address  
2598 4238 1 is passed here so that we can both  
2599 4239 1 use and update this pointer.  
2600 4240 1 OUT_PC_PTR -The address of a pointer to where we  
2601 4241 1 will eventually be stuffing the encoded  
2602 4242 1 instruction in memory. We both read  
2603 4243 1 and write (update) this pointer.  
2604 4244 1 INST_CS_PTR -The address of a pointer to the counted  
2605 4245 1 string which describes where we are at  
2606 4246 1 in operand encoding. Again, a pointer is  
2607 4247 1 passed here so that we may update the cs-pointer  
2608 4248 1 CTRL_STRING -A 4-character 'string' (a longword) which effectively  
2609 4249 1 controls the action taken by this routine. The first  
2610 4250 1 (0th) character should be 'Y' or 'N', and is taken to  
2611 4251 1 indicate whether we should allow indexing  
2612 4252 1 for the current operand reference or not.  
2613 4253 1 The next 2 or 3 characters must be 1 of 'B', 'C', or 'D',  
2614 4254 1 and are used to indicate how the remaining parameters  
2615 4255 1 should be interpreted. See below.  
2616 4256 1 The last of these characters must be 0 (null) to  
2617 4257 1 indicate when the routine should stop.  
2618 4258 1 ARG? -These args are interpreted differently depending  
2619 4259 1 on the 'control string'. See above and below.  
2620 4260 1  
2621 4261 1 Implicit Inputs:  
2622 4262 1  
2623 4263 1 None.  
2624 4264 1  
2625 4265 1 Outputs:  
2626 4266 1  
2627 4267 1 None.
```

```
: 2628 4268 1 |
: 2629 4269 1 | Implicit Outputs:
: 2630 4270 1 |
: 2631 4271 1 |     The instruction bytes are copied into the output vector.
: 2632 4272 1 |
: 2633 4273 1 | Routine Value:
: 2634 4274 1 |
: 2635 4275 1 |     TRUE - if all went OK,
: 2636 4276 1 |     FALSE otherwise. The only thing that can go wrong
: 2637 4277 1 |     is that we are prepared to allow indexing, see that the
: 2638 4278 1 |     indexing reference is started, '[' is encountered),
: 2639 4279 1 |     but then don't get a proper completion of this token.
: 2640 4280 1 | --
: 2641 4281 1 |
: 2642 4282 2 | BEGIN
: 2643 4283 2 |
: 2644 4284 2 | MAP
: 2645 4285 2 |     ++
: 2646 4286 2 |     The reason why the following 3 are REFs to LONGs instead of to BYTEs
: 2647 4287 2 |     is because they are actually REF REF VECTOR[,BYTE], which we can only
: 2648 4288 2 |     achieve (so far!?) via a REF LONG.
: 2649 4289 2 |     --
: 2650 4290 2 |     OUT_PC_PTR : REF VECTOR[,LONG],
: 2651 4291 2 |     INST_STG_DESC : REF BLOCK[,BYTE],
: 2652 4292 2 |     OUT_BYTE_STREAM : REF VECTOR[,LONG];
: 2653 4293 2 |
: 2654 4294 2 | LOCAL
: 2655 4295 2 |     CTRL_PTR : REF VECTOR[,BYTE],
: 2656 4296 2 |     OUT_BYTE_PTR : REF VECTOR[,BYTE],
: 2657 4297 2 |     ARG_PTR : REF VECTOR[,LONG],
: 2658 4298 2 |     TOKEN_BUFFER : VECTOR[CHS_PER_LEXEME, BYTE],
: 2659 4299 2 |     TOKEN_STG_DESC : BLOCK[12, BYTE];
: 2660 4300 2 |
: 2661 4301 2 |     ++
: 2662 4302 2 |     Set up the various pointers. The ARG_PTR is used to access each successive
: 2663 4303 2 |     ARGx actual parameter. Since this routine loops thru them, it can't use the
: 2664 4304 2 |     formal parameter's name.
: 2665 4305 2 |     --
: 2666 4306 2 |     ARG_PTR = ARG0;
: 2667 4307 2 |
: 2668 4308 2 |     ++
: 2669 4309 2 |     CTRL_PTR points to the individual characters passed in the actual parameter,
: 2670 4310 2 |     'CTRL_STRING'. Note that this is actually a literal, because the characters
: 2671 4311 2 |     are contained within the parameter.
: 2672 4312 2 |     --
: 2673 4313 2 |     CTRL_PTR = CTRL_STRING;
: 2674 4314 2 |
: 2675 4315 2 |     ++
: 2676 4316 2 |     OUT_BYTE_PTR points to the location for the next byte in the instruction
: 2677 4317 2 |     stream to be written. This value is found by loading the contents of the cell
: 2678 4318 2 |     pointed to by OUT_BYTE_STREAM. The address of this pointer must be passed to
: 2679 4319 2 |     enable the pointer to be incremented.
: 2680 4320 2 |     --
: 2681 4321 2 |     OUT_BYTE_PTR = .OUT_BYTE_STREAM[0];
: 2682 4322 2 |
: 2683 4323 2 |     ++
: 2684 4324 2 |     Likewise, the address of the current instruction-stream counted-string
```

```
2685 4325 2 | pointer, INST_CS, is passed. Once again this provides the ability to update
2686 4326 2 | the pointer.
2687 4327 2 |--
2688 4328 2 |TOKEN_STG_DESC [DSC$W_LENGTH] = 0;
2689 4329 2 |TOKEN_STG_DESC [DSC$A_POINTER] = TOKEN_BUFFER;
2690 4330 2 |TOKEN_STG_DESC [DSC$W_MAXLEN] = CHS_PER_LEXEME;
2691 4331 2 |
2692 4332 2 |++
2693 4333 2 | Whether or not indexing mode is allowed, is indicated by the first character
2694 4334 2 | in the control string. 'Y' means yes, and anything else means no.
2695 4335 2 |--
2696 4336 3 |IF (.CTRL_PTR[0] EQL 'Y')
2697 4337 2 |THEN
2698 4338 3 |   BEGIN
2699 4339 3 |       ++
2700 4340 3 |       Check for indexing. The difficulty here is that this routine must
2701 4341 3 |       'look ahead' before calling GET_NEXT_TOKEN because it will overwrite
2702 4342 3 |       the instruction string. The string must be left untouched unless
2703 4343 3 |       indexing mode was actually specified.
2704 4344 3 |       --
2705 4345 3 |       LOCAL
2706 4346 3 |           INPUT_PTR,
2707 4347 3 |           CHAR;
2708 4348 3 |
2709 4349 3 |       INPUT_PTR = .INST_STG_DESC [DSC$A_POINTER];
2710 4350 3 |       CHAR = CH$RCHAR (.INPUT_PTR);
2711 4351 4 |       IF (.CHAR EQL '[')
2712 4352 3 |       THEN
2713 4353 4 |           BEGIN
2714 4354 4 |               ++
2715 4355 4 |               Error if this is not an indexed reference.
2716 4356 4 |               --
2717 4357 5 |               IF (GET_NEXT_TOKEN(.INST_STG_DESC, TOKEN_BUFFER) NEQ INDEXING_TOKEN)
2718 4358 4 |               THEN
2719 4359 5 |                   BEGIN
2720 4360 5 |                       SIGNAL(PAT$ OPSYNTAX+MSG$K_INFO);
2721 4361 5 |                       RETURN(FALSE);
2722 4362 4 |                   END;
2723 4363 4 |
2724 4364 4 |               ++
2725 4365 4 |               Output the indexing mode byte.
2726 4366 4 |               --
2727 4367 4 |               OUT_BYTE_PTR[0] = (INDEXING_MODE ^ 4) OR .TOKEN_BUFFER[0];
2728 4368 4 |               OUT_BYTE_PTR = .OUT_BYTE_PTR + BYTE_LENGTH;
2729 4369 3 |           END;
2730 4370 2 |       END;
2731 4371 2 |
2732 4372 2 |       ++
2733 4373 2 |       Continue on according to the control string, breaking out of the loop when
2734 4374 2 |       the first 0 byte is encountered.
2735 4375 2 |       --
2736 4376 2 |       CTRL_PTR = .CTRL_PTR + 1;
2737 4377 2 |   DO
2738 4378 3 |       BEGIN
2739 4379 3 |       BIND
2740 4380 3 |           ARG_BYTE      = (.ARG_PTR) : REF VECTOR[.BYTE];
2741 4381 3 |
```

```

: 2742      4382 3      SELECTONE .CTRL_PTR[0] OF
: 2743      4383 3      SET
: 2744      4384 3
: 2745      4385 3      ['B']: ! Pass back 1 byte.
: 2746      4386 3
: 2747      4387 4      BEGIN
: 2748      4388 4      OUT_BYTE_PTR[0] = .ARG_PTR[0];
: 2749      4389 4      OUT_BYTE_PTR = .OUT_BYTE_PTR + BYTE_LENGTH;
: 2750      4390 4      ARG_PTR = .ARG_PTR + LONG_LENGTH;
: 2751      4391 3      END;
: 2752      4392 3
: 2753      4393 3      ['C']: ! Counted byte string.
: 2754      4394 3
: 2755      4395 4      BEGIN
: 2756      4396 4      CH$MOVE(.ARG_BYTE[0], ARG_BYTE[1], .OUT_BYTE_PTR);
: 2757      4397 4      OUT_BYTE_PTR = .OUT_BYTE_PTR + .ARG_BYTE[0];
: 2758      4398 4      ARG_PTR = .ARG_PTR + LONG_LENGTH;
: 2759      4399 3      END;
: 2760      4400 3
: 2761      4401 3      ['D']: ! Count + byte address.
: 2762      4402 3
: 2763      4403 4      BEGIN
: 2764      4404 4      CH$MOVE(.ARG_PTR[0], .ARG_PTR[1], .OUT_BYTE_PTR);
: 2765      4405 4      OUT_BYTE_PTR = .OUT_BYTE_PTR + .ARG_PTR[0];
: 2766      4406 4      ARG_PTR = .ARG_PTR + 2*LONG_LENGTH;
: 2767      4407 3      END;
: 2768      4408 3
: 2769      4409 3      [OTHERWISE]: ! Error.
: 2770      4410 3      RETURN(0);
: 2771      4411 3
: 2772      4412 3      TES;
: 2773      4413 3
: 2774      4414 3      !++
: 2775      4415 3      ! Loop back to consider the next control character
: 2776      4416 3      ! until a null character is encountered.
: 2777      4417 3      !--
: 2778      4418 3      CTRL_PTR = .CTRL_PTR + 1;
: 2779      4419 3      END
: 2780      4420 3
: 2781      4421 2      WHILE (.CTRL_PTR[0] NEQ 0);
: 2782      4422 2
: 2783      4423 2      !++
: 2784      4424 2      ! Update the instruction-stream pointer which is being maintained by the routine
: 2785      4425 2      ! which called this one. This can be done as the address of the pointer was
: 2786      4426 2      ! passed. Also, this is possible this routine remembers the number of bytes it
: 2787      4427 2      ! has written to the output byte stream.
: 2788      4428 2      !--
: 2789      4429 2      OUT_PC_PTR[0] = .OUT_PC_PTR[0] + (.OUT_BYTE_PTR - .OUT_BYTE_STREAM[0]);
: 2790      4430 2      OUT_BYTE_STREAM[0] = .OUT_BYTE_PTR;
: 2791      4431 2
: 2792      4432 2      !++
: 2793      4433 2      ! Likewise, update the caller's instruction string pointer.
: 2794      4434 2      !--
: 2795      4435 2      RETURN(TRUE);
: 2796      4436 1      END;
```



```
03FC 00000 INST_OUTPUT:
      5E      28 C2 00002      .WORD      Save R2,R3,R4,R5,R6,R7,R8,R9      : 4211
      58      14 AC 9E 00005      SUBL2      #40, SP      :
      59      10 AC 9E 00009      MOVAB      ARG0, ARG_PTR      : 4306
      56      04 BC D0 0000D      MOVAB      CTRL_STRING, CTRL_PTR      : 4313
      04 AE      0C AE 9E 00013      MOVAB      @OUT_BYTE_STREAM, OUT_BYTE_PTR      : 4321
      08 AE      19 B0 00018      CLRW      TOKEN_STG_DESC      : 4328
      59 8F      69 91 0001C      MOVAB      TOKEN_BUFFER, TOKEN_STG_DESC+4      : 4329
      3C 12 00020      MOVW      #25, TOKEN_STG_DESC+8      : 4330
      50      0C AC D0 00022      CMPB      (CTRL_PTR), #89      : 4336
      51      04 A0 D0 00026      BNEQ      2$      :
      51      61 9A 0002A      MOVL      INST_STG_DESC, R0      : 4349
      8F      51 D1 0002D      MOVL      4(R0), INPUT_PTR      :
      0000005B 8F      28 12 00034      MOVZBL   (INPUT_PTR), CHAR      : 4350
      3C 12 00034      CMPL      CHAR, #91      : 4351
      0C AE 9F 00036      BNEQ      2$      :
      50 DD 00039      PUSHAB   TOKEN_BUFFER      : 4357
      F1ED CF      50 DD 00039      PUSHL      R0      :
      000000F0 8F      02 FB 0003B      CALLS     #2, GET_NEXT_TOKEN      :
      50 D1 00040      CMPL      R0, #240      :
      0F 13 00047      BEQL      1$      :
      00000000G 00 006D821B 8F DD 00049      PUSHL      #7176731      : 4360
      01 FB 0004F      CALLS     #1, LIB$SIGNAL      :
      54 11 00056      BRB      7$      : 4361
      86 0C AE      40 8F 89 00058 1$: BISB3     #64, TOKEN_BUFFER, (OUT_BYTE_PTR)+      : 4367
      59 D6 0005E 2$: INCL      CTRL_PTR      : 4376
      42 8F      69 91 00060 3$: CMPB      (CTRL_PTR), #66      : 4385
      05 12 00064      BNEQ      4$      :
      86      68 90 00066      MOVB      (ARG_PTR), (OUT_BYTE_PTR)+      : 4388
      27 11 00069      BRB      6$      : 4390
      43 8F      69 91 0006B 4$: CMPB      (CTRL_PTR), #67      : 4393
      13 12 0006F      BNEQ      5$      :
      57      68 D0 00071      MOVL      (ARG_PTR), R7      : 4396
      50      67 9A 00074      MOVZBL   (R7), R0      :
      66 01 A7      50 28 00077      MOV(C3)   R0, 1(R7), (OUT_BYTE_PTR)      :
      50      67 9A 0007C      MOVZBL   (R7), R0      : 4397
      56      50 C0 0007F      ADDL2     R0, OUT_BYTE_PTR      :
      0E 11 00082      BRB      6$      : 4398
      44 8F      69 91 00084 5$: CMPB      (CTRL_PTR), #68      : 4401
      22 12 00088      BNEQ      7$      :
      66 04 B8      68 28 0008A      MOV(C3)   (ARG_PTR), @4(ARG_PTR), (OUT_BYTE_PTR)      : 4404
      56      88 C0 0008F      ADDL2     (ARG_PTR)+, OUT_BYTE_PTR      : 4405
      58      04 C0 00092 6$: ADDL2     #4, ARG_PTR      : 4406
      59 D6 00095      INCL      CTRL_PTR      : 4418
      69 95 00097      TSTB      (CTRL_PTR)      : 4421
      C5 12 00099      BNEQ      3$      :
      50      56      04 BC C3 0009B      SUBL3     @OUT_BYTE_STREAM, OUT_BYTE_PTR, R0      : 4429
      08 BC      50 C0 000A0      ADDL2     R0, @OUT_PC_PTR      :
      04 BC      56 D0 000A4      MOVL      OUT_BYTE_PTR, @OUT_BYTE_STREAM      : 4430
      50      01 D0 000A8      MOVL      #1, R0      : 4435
      04 000AB      RET      :
      50 D4 000AC 7$: CLRL      R0      : 4436
      04 000AE      RET      :
```

PATENC
V04-000

C 1
16-Sep-1984 00:40:15
14-Sep-1984 12:52:32

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[PATCH.SRC]PATENC.B32;1 (10) Page 80

; Routine Size: 175 bytes, Routine Base: _PAT\$CODE + 105F

PAT
V04

; R

```

: 2798 4437 1 ROUTINE OPCODE_MATCH (OPCO_STG_DESC, OPINFO_PTR) =
: 2799 4438 1
: 2800 4439 1 ++
: 2801 4440 1 Functional Description:
: 2802 4441 1
: 2803 4442 1 Look up the given opcode mnemonic in the OPINFO
: 2804 4443 1 table to see if it is there. If so, return the
: 2805 4444 1 opcode and a pointer to the OPINFO table entry which
: 2806 4445 1 corresponds to this opcode.
: 2807 4446 1
: 2808 4447 1 Calling Sequence:
: 2809 4448 1
: 2810 4449 1 OPCODE_MATCH ()
: 2811 4450 1
: 2812 4451 1 Inputs:
: 2813 4452 1
: 2814 4453 1 OPCO_STG_DESC -String descriptor for the given opcode.
: 2815 4454 1 OPINFO_PTR -The address of where we should copy the
: 2816 4455 1 OPINFO record pointer when we find the
: 2817 4456 1 one which corresponds to the given opcode.
: 2818 4457 1 Implicit Inputs:
: 2819 4458 1
: 2820 4459 1 The OPINFO table. See PATINS.B32
: 2821 4460 1
: 2822 4461 1 Outputs:
: 2823 4462 1
: 2824 4463 1 None.
: 2825 4464 1
: 2826 4465 1 Implicit Outputs:
: 2827 4466 1
: 2828 4467 1 A pointer to the OPINFO table entry that
: 2829 4468 1 corresponds to the found opcode mnemonic
: 2830 4469 1 is returned via the formal pointer OPINFO_PTR.
: 2831 4470 1
: 2832 4471 1 Returned Value:
: 2833 4472 1
: 2834 4473 1 -1 -if the lookup fails or if there is insufficient
: 2835 4474 1 information in the table entry for the program
: 2836 4475 1 to continue.
: 2837 4476 1 The found OPCODE, otherwise. This is non-standard,
: 2838 4477 1 so we use a local macro to draw attention to it.
: 2839 4478 1 --
: 2840 4479 1
: 2841 4480 2 BEGIN
: 2842 4481 2
: 2843 4482 2 MACRO
: 2844 4483 2 OPC_MATCH_ERROR = -1 %; ! Error return for this routine.
: 2845 4484 2
: 2846 4485 2 MAP
: 2847 4486 2 OPCO_STG_DESC : REF BLOCK [, BYTE],
: 2848 4487 2 OPINFO_PTR : REF VECTOR[,LONG];
: 2849 4488 2
: 2850 4489 2 LOCAL
: 2851 4490 2 ALIAS,
: 2852 4491 2 OPCODE,
: 2853 4492 2 OP_SIZE,
: 2854 4493 2 OP_RAD50.
```

```

: 2855      4494 2      OP_FROM_USER      :      VECTOR[ OP_CH_SIZE, BYTE ];
: 2856      4495 2
: 2857      4496 2      ++
: 2858      4497 2      First check that the supposed opcode is not too long or too short.
: 2859      4498 2      A zero cannot be returned as zero is a valid opcode.
: 2860      4499 2      --
: 2861      4500 2      OP_SIZE = .OPCO_STG_DESC [DSC$W_LENGTH];
: 2862      4501 3      IF (.OP_SIZE GTR OP_CH_SIZE) OR (.OP_SIZE LEQ 0)
: 2863      4502 2      THEN
: 2864      4503 2          RETURN(OPC_MATCH_ERROR);
: 2865      4504 2
: 2866      4505 2      ++
: 2867      4506 2      Otherwise do the lookup linearly by first filling a local vector with spaces,
: 2868      4507 2      copying in the given opcode mnemonic, and converting to RAD50 so longword
: 2869      4508 2      compares can be used with the Opcodes encoded in the table.
: 2870      4509 2      --
: 2871      4510 2      CH$COPY(.OP_SIZE, CH$PTR(.OPCO_STG_DESC[DSC$A_POINTER], 0), ASC_SPACE,
: 2872      4511 2          OP_CH_SIZE, OP_FROM_USER);
: 2873      4512 2      OP_RAD50 = RAD50( OP_FROM_USER );
: 2874      4513 2      INCR OPCODE FROM 0 TO SIZEOPINFO1-1 DO
: 2875      4514 3          BEGIN
: 2876      4515 3              ++
: 2877      4516 3              Extract the opcode from the OPINFO table, converting it to ASCII, and
: 2878      4517 3              compare this with what is being sought.
: 2879      4518 3              --
: 2880      4519 3              IF .PAT$GB_OPINFO1[.OPCODE, OP_NAME] EQL .OP_RAD50
: 2881      4520 3              THEN
: 2882      4521 4                  BEGIN
: 2883      4522 4                      ++
: 2884      4523 4                      Pass back both the opcode (the current index into the table),
: 2885      4524 4                      and the pointer to the current table entry.
: 2886      4525 4                      --
: 2887      4526 4                      OPINFO_PTR[0] = PAT$GB_OPINFO1[.OPCODE, OP_NAME ];
: 2888      4527 4                      RETURN(.OPCODE);
: 2889      4528 3                  END;
: 2890      4529 2          END;
: 2891      4530 2      INCR OPCODE FROM 0 TO SIZEOPINFO2-1 DO
: 2892      4531 3          BEGIN
: 2893      4532 3              ++
: 2894      4533 3              Extract the opcode from the OPINFO table, converting it to ASCII, and
: 2895      4534 3              compare this with what is being sought.
: 2896      4535 3              --
: 2897      4536 3              IF .PAT$GB_OPINFO2[.OPCODE, OP_NAME] EQL .OP_RAD50
: 2898      4537 3              THEN
: 2899      4538 4                  BEGIN
: 2900      4539 4                      ++
: 2901      4540 4                      Pass back both the opcode (the current index into the table),
: 2902      4541 4                      and the pointer to the current table entry.
: 2903      4542 4                      --
: 2904      4543 4                      OPINFO_PTR[0] = PAT$GB_OPINFO2[.OPCODE, OP_NAME ];
: 2905      4544 4                      RETURN(.OPCODE*8+X'FD');
: 2906      4545 4                  END;
: 2907      4546 3              END;
: 2908      4547 2          END;
: 2909      4548 2      INCR ALIAS FROM 0 TO SIZEALIAS-1 DO
: 2910      4549 3          BEGIN
: 2911      4550 3
```

```

: 2912      4551 3      :++
: 2913      4552 3      : Extract the opcode from the ALIAS table, converting it to ASCII, and
: 2914      4553 3      : compare this with what is being sought.
: 2915      4554 3      :--
: 2916      4555 3      IF .PAT$GB_ALIAS[.ALIAS, OP_NAME] EQL .OP_RAD50
: 2917      4556 3      THEN
: 2918      4557 4          BEGIN
: 2919      4558 4              :++
: 2920      4559 4              : Pass back both the opcode (the current index into the table),
: 2921      4560 4              : and the pointer to the current table entry.
: 2922      4561 4              :--
: 2923      4562 4              OPCODE = .PAT$GB_ALIAS[.ALIAS, AL OPC ];
: 2924      4563 4              OPINFO_PTR[0] = PAT$GB_OPINFO[.OPCODE, OP_NAME ];
: 2925      4564 4              RETURN(.OPCODE);
: 2926      4565 3          END;
: 2927      4566 2      END;
: 2928      4567 2
: 2929      4568 2      :++
: 2930      4569 2      : Failure, if routine falls through loop.
: 2931      4570 2      :--
: 2932      4571 2      RETURN(OPC_MATCH_ERROR);
: 2933      4572 1      END;
```

```

                                00FC 00000 OPCODE_MATCH:
                                .WORD      Save R2,R3,R4,R5,R6,R7
                                57 00000000G EF 9E 00002      MOVAB      PAT$GB_OPINFO2, R7
                                56 00000000G EF 9E 00009      MOVAB      PAT$GB_OPINFO1, R6
                                5E          08 C2 00010      SUBL2      #8, SP
                                50          04 AC D0 00013      MOVL      OPCO_STG_DESC, R0
                                51          60 3C 00017      MOVZWL     (R0), OP_SIZE
                                06          51 D1 0001A      CMPL      OP_SIZE, #6
                                03          15 0001D      BLEQ      2$
                                008E        31 0001F 1$:      BRW       12$
                                51          D5 00022 2$:      TSTL      OP_SIZE
                                F9          15 00024      BLEQ      1$
                                06          51 2C 00026      MOVCS     OP_SIZE, @4(R0), #32, #6, OP_FROM_USER
                                6E          0002C
                                5E          DD 0002D      PUSHL     SP
                                00000000G EF 01 FB 0002F      CALLS     #1, RAD50
                                51          D4 00036      CLRL      OPCODE
                                6641        7F 00038 3$:      PUSHAQ    PAT$GB_OPINFO1[OPCODE]
                                50          9E D1 0003B      CMPL      @ (SP)+, OP_RAD50
                                09          12 0003E      BNEQ      4$
                                08          BC 6641 7E 00040      MOVAQ     PAT$GB_OPINFO1[OPCODE], @OPINFO_PTR
                                50          51 D0 00045      MOVL      OPCODE, R0
                                E7          51 00000102 8F F3 00049 4$:      AOBLEQ    #258, OPCODE, 3$
                                51          D4 00051      CLRL      OPCODE
                                6741        7F 00053 5$:      PUSHAQ    PAT$GB_OPINFO2[OPCODE]
                                50          9E D1 00056      CMPL      @ (SP)+, OP_RAD50
                                10          12 00059      BNEQ      6$
                                08          BC 6741 7E 0005B      MOVAQ     PAT$GB_OPINFO2[OPCODE], @OPINFO_PTR
                                52          51 08 78 00060      ASHL      #8, OPCODE, R2
```

	52	00FD	C2	9E	00064	MOVAB	253(R2), R2	:	
			3D	1	00069	BRB	10\$	:	
E0	51	000000FF	8F	13	0006B 6\$:	AOBLEQ	#255, OP CODE, 5\$	:	4531
			51	C4	00073	CLRL	ALIAS	:	4549
53	51		06	C5	00075 7\$:	MULL3	#6, ALIAS, R3	:	4555
		00000000GEF	43	9F	00079	PUSHAB	PAT\$GB_ALIAS[R3]	:	
	50		9E	D1	00080	CMPL	@(SP)+, OP_RAD50	:	
			27	12	00083	BNEQ	11\$	:	
		00000000GEF	43	9F	00085	PUSHAB	PAT\$GB_ALIAS+4[R3]	:	4562
	52		9E	3C	0008C	MOVZWL	@(SP)+, OP CODE	:	
FD	8F		52	91	0008F	CMPB	OP CODE, #253	:	4563
			06	13	00093	BEQL	8\$	:	
	53		66	7E	00095	MOVAQ	PAT\$GB_OPINFO1[OP CODE], R3	:	
			09	11	00099	BRB	9\$	:	
53	52	F8	8F	78	0009B 8\$:	ASHL	#-8, OP CODE, R3	:	
	53		67	7E	000A0	MOVAQ	PAT\$GB_OPINFO2[R3], R3	:	
	08	BC	53	D0	000A4 9\$:	MOVL	R3, @OPINFO_PTR	:	
	50		52	D0	000A8 10\$:	MOVL	OP CODE, R0	:	4564
				04	000AB	RET		:	
C5	51		34	F3	000AC 11\$:	AOBLEQ	#52, ALIAS, 7\$	:	4549
	50		01	CF	000B0 12\$:	MNEGL	#1, R0	:	4571
			04	000B3	RET			:	4572

; Routine Size: 180 bytes, Routine Base: _PAT\$CODE + 110E

```

: 2935 4573 1 ROUTINE MAR_GET_LEX (INPUT_STG_DESC, LEXEME_STG_DESC) =
: 2936 4574 1
: 2937 4575 1 !++
: 2938 4576 1 Functional Description:
: 2939 4577 1
: 2940 4578 1 Extracts a lexeme from the input stream by calling the routine
: 2941 4579 1 PAT$MAR_GET_LEX. First MAR_GET_LEX zeros the lexeme buffer.
: 2942 4580 1 The value of the routine is the token in the character string
: 2943 4581 1 pointed to by LEXEME_STG_DESC, the string descriptor for the lexeme.
: 2944 4582 1 Also the input buffer string descriptor is updated.
: 2945 4583 1
: 2946 4584 1 Calling Sequence:
: 2947 4585 1
: 2948 4586 1 MAR_GET_LEX ( INPUT_STG_DESC, LEXEME_STG_DESC)
: 2949 4587 1
: 2950 4588 1 Inputs:
: 2951 4589 1
: 2952 4590 1 INPUT_STG_DESC - String descriptor for input line
: 2953 4591 1 LEXEME_STG_DESC - String descriptor for lexeme found
: 2954 4592 1
: 2955 4593 1 Implicit Inputs:
: 2956 4594 1
: 2957 4595 1 NONE
: 2958 4596 1
: 2959 4597 1 Outputs:
: 2960 4598 1
: 2961 4599 1 None.
: 2962 4600 1
: 2963 4601 1 Implicit Outputs:
: 2964 4602 1
: 2965 4603 1 The string descriptors are updated.
: 2966 4604 1
: 2967 4605 1 Returned Value:
: 2968 4606 1
: 2969 4607 1 An encoded representation of the token found.
: 2970 4608 1
: 2971 4609 1 --
: 2972 4610 1
: 2973 4611 2 BEGIN
: 2974 4612 2
: 2975 4613 2 MAP
: 2976 4614 2 INPUT_STG_DESC : REF BLOCK [, BYTE],
: 2977 4615 2 LEXEME_STG_DESC : REF BLOCK[,BYTE];
: 2978 4616 2
: 2979 4617 2 !++
: 2980 4618 2 First zero out the buffer which will hold the lexeme found.
: 2981 4619 2 --
: 2982 4620 2 ZEROCOR (.LEXEME_STG_DESC[DSC$A_POINTER], (.LEXEME_STG_DESC[DSC$W_MAXLEN]/4));
: 2983 4621 2 RETURN(PAT$MAR_GET_LEX(.INPUT_STG_DESC, .LEXEME_STG_DESC));
: 2984 4622 1 END;
```

007C 00000 MAR_GET_LEX:
.WORD Save R2,R3,R4,R5,R6

: 4573

PATENC
V04-000

I 1
16-Sep-1984 00:40:15
14-Sep-1984 12:52:32

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[PATCH.SRC]PATENC.B32;1 (12)

Page 86

**

56	08	AC	D0	00002	MOVL	LEXEME_STG_DESC, R6	:	4620
50	08	A6	3C	00006	MOVZWL	8(R6), R0	:	
50		04	C6	0C00A	DIVL2	#4, R0	:	
50		04	C4	0000D	MULL2	#4, R0	:	
6E		00	2C	00010	MOVCS	#0, (SP), #0, R0, @4(R6)	:	
	04	B6		00015			:	
		56	DD	00017	PUSHL	R6	:	4621
	04	AC	DD	00019	PUSHL	INPUT_STG_DESC	:	
00000000G	EF	02	FB	0001C	CALLS	#2, PAT\$MAR_GET_LEX	:	
		04	00	00023	RET		:	4622

; Routine Size: 36 bytes, Routine Base: _PAT\$CODE + 11C2


```

: 2986 4623 1 ROUTINE ADD_LEX_T_OPRND (LEX_STG_DESC, OPRND_STG_DESC, MAX_BUF_SIZ) : NOVALUE =
: 2987 4624 1
: 2988 4625 1 |++
: 2989 4626 1 | Functional Description:
: 2990 4627 1
: 2991 4628 1 |     This routine takes an ASCII lexeme and adds it to an operand buffer.
: 2992 4629 1 |     It may be used to take any ASCII string described by a string
: 2993 4630 1 |     descriptor and add it to a buffer described by another string
: 2994 4631 1 |     descriptor. The string descriptor for the latter is updated to
: 2995 4632 1 |     include the new ASCII string. The third input parameter gives the
: 2996 4633 1 |     maximum size of the operand buffer.
: 2997 4634 1
: 2998 4635 1 | Calling Sequence:
: 2999 4636 1
: 3000 4637 1 |     ADD_LEX_T_OPRND ( LEX_STG_DESC, OPRND_STG_DESC)
: 3001 4638 1
: 3002 4639 1 | Inputs:
: 3003 4640 1
: 3004 4641 1 |     LEX_STG_DESC - String descriptor for input lexeme
: 3005 4642 1 |     OPRND_STG_DESC - String descriptor for resultant operand string
: 3006 4643 1 |     MAX_BUF_SIZ - Size of the buffer pointed to by OPRND_STG_DESC
: 3007 4644 1
: 3008 4645 1 | Implicit Inputs:
: 3009 4646 1
: 3010 4647 1 |     NONE
: 3011 4648 1
: 3012 4649 1 | Outputs:
: 3013 4650 1
: 3014 4651 1 |     None.
: 3015 4652 1
: 3016 4653 1 | Implicit Outputs:
: 3017 4654 1
: 3018 4655 1 |     The lexeme string is written into the next free bytes of the operand
: 3019 4656 1 |     buffer. The string descriptor for the resultant string, OPRND_STG_DESC,
: 3020 4657 1 |     is updated. If the combined string is too large for the buffer
: 3021 4658 1 |     then an error is SIGNALed.
: 3022 4659 1
: 3023 4660 1 | Returned Value:
: 3024 4661 1
: 3025 4662 1 |     none
: 3026 4663 1
: 3027 4664 1 | --
: 3028 4665 1
: 3029 4666 2 BEGIN
: 3030 4667 2
: 3031 4668 2 MAP
: 3032 4669 2     LEX_STG_DESC : REF BLOCK [, BYTE],
: 3033 4670 2     OPRND_STG_DESC : REF BLOCK[, BYTE];
: 3034 4671 2
: 3035 4672 2 |++
: 3036 4673 2 | First check that there is room in the operand buffer for the lexeme string.
: 3037 4674 2 | --
: 3038 4675 3 IF ((.LEX_STG_DESC[DSC$W_LENGTH] + .OPRND_STG_DESC[DSC$W_LENGTH]) GTR .OPRND_STG_DESC[DSC$W_MAXLEN])
: 3039 4676 2 THEN
: 3040 4677 2     SIGNAL(PAT$ OPRNDLNG+MSG$K WARN, 4, .OPRND_STG_DESC[DSC$W_LENGTH],
: 3041 4678 2         .OPRND_STG_DESC[DSC$A_POINTER], .LEX_STG_DESC[DSC$W_LENGTH],
: 3042 4679 2         .LEX_STG_DESC[DSC$A_POINTER]);
```

```

: 3043      4680  2
: 3044      4681  2  !++
: 3045      4682  2  ! Plenty of room in buffer.  Move the lexeme string into the buffer, starting
: 3046      4683  2  ! at the next unused byte in the buffer.
: 3047      4684  2  !--
: 3048      4685  2  CH$MOVE(.LEX_STG_DESC[DSC$W_LENGTH], .LEX_STG_DESC[DSC$A_POINTER],
: 3049      4686  2  (CH$PTR(.OPRND_STG_DESC[DSC$A_POINTER], .OPRND_STG_DESC[DSC$W_LENGTH]));
: 3050      4687  2
: 3051      4688  2  !++
: 3052      4689  2  ! Now update the operand string descriptor to include the appended lexeme.
: 3053      4690  2  !--
: 3054      4691  2  OPRND_STG_DESC[DSC$W_LENGTH] = .OPRND_STG_DESC[DSC$W_LENGTH] + .LEX_STG_DESC[DSC$W_LENGTH];
: 3055      4692  2  RETUP;
: 3056      4693  1  END;
```

```

                                00FC 00000 ADD_LEX_T OPRND:
                                .WORD      Save R2,R3,R4,R5,R6,R7
                                MOVL      LEX_STG_DESC, R7
                                MOVL      OPRND_STG_DESC, R6
                                MOVZWL   (R7), -R0
                                MOVZWL   (R6), R1
                                ADDL2    R1, R0
                                CMPZV    #0, #16, 8(R6), R0
                                BGEQ     1$
                                PUSHL    4(R7)
                                MOVZWL   (R7), -(SP)
                                PUSHL    4(R6)
                                MOVZWL   (R6), -(SP)
                                PUSHL    #4
                                PUSHL    #7176856
                                CALLS    #6, LIB$SIGNAL
                                MOVZWL   (R6), R0
                                ADDL2    4(R6), R0
                                MOVC3    (R7), @4(R7), (R0)
                                ADDW2    (R7), (R6)
                                RET
                                1$:
                                57      04  AC  DO 00002
                                56      08  AC  DO 00006
                                50      67  3C 0000A
                                51      66  3C 0000D
                                50      51  C0 00010
                                10      00  ED 00013
                                18      18  00019
                                04      A7  DD 0001B
                                7E      67  3C 0001E
                                04      A6  DD 00021
                                7E      66  3C 00024
                                04      DD 00027
                                006D8298 8F  DD 00029
                                000C0000G 00  06  FB 0002F
                                50      66  3C 00036 1$:
                                50      A6  C0 00039
                                60      04  87  28 0003D
                                66      67  A0 00042
                                04      04  00045
```

; Routine Size: 70 bytes, Routine Base: _PAT\$CODE + 11E6

```
3058 4694 1 GLOBAL ROUTINE PAT$REDUCE_INS (INS_PTR, EDITED_INS_DESC) : NOVALUE =
3059 4695 1
3060 4696 1 !**
3061 4697 1 FUNCTIONAL DESCRIPTION:
3062 4698 1
3063 4699 1 This routine takes an ASCII symbolic instruction and outputs a
3064 4700 1 reduced ASCII instruction, that is, one with expressions reduced to
3065 4701 1 single values and symbolic names replaced with values. The resultant
3066 4702 1 instruction is written into the buffer pointed to by EDITED_INS_DESC.
3067 4703 1 The string descriptor is updated to contain the edited length.
3068 4704 1
3069 4705 1 FORMAL PARAMETERS:
3070 4706 1
3071 4707 1 INS_PTR - Pointer to the symbolic instruction to be reduced
3072 4708 1 EDITED_INS_DESC - String descriptor for output buffer
3073 4709 1
3074 4710 1 IMPLICIT INPUTS:
3075 4711 1
3076 4712 1 EDITED_INS_DESC is already initialized.
3077 4713 1
3078 4714 1 IMPLICIT OUTPUTS:
3079 4715 1
3080 4716 1 The reduced instruction is written into the output buffer.
3081 4717 1
3082 4718 1 ROUTINE VALUE:
3083 4719 1
3084 4720 1 none
3085 4721 1
3086 4722 1 COMPLETION CODES:
3087 4723 1
3088 4724 1 NONE
3089 4725 1
3090 4726 1 SIDE EFFECTS:
3091 4727 1
3092 4728 1 If the reduction could not be performed, then an error message is
3093 4729 1 SIGNALed and an UNWIND is performed.
3094 4730 1 EDITED_INS_DESC contains an updated length.
3095 4731 1
3096 4732 1 --
3097 4733 1
3098 4734 2 BEGIN
3099 4735 2
3100 4736 2 MAP
3101 4737 2 EDITED_INS_DESC : REF BLOCK[,BYTE], ! String descriptor for reduced instruction
3102 4738 2 INS_PTR : REF VECTOR[,BYTE]; ! Pointer to input instruction (ASCII)
3103 4739 2 LOCAL
3104 4740 2 CHAR : BYTE, ! Next character in instruction string
3105 4741 2 INSTRUCT_PTR : REF VECTOR[,BYTE], ! Local pointer into instruction
3106 4742 2 LABEL_FLAG, ! Label flag
3107 4743 2 INS_STG_DESC : BLOCK[8,BYTE], ! String descriptor for non-reduced part of
3108 4744 2 LEXEME_DESC : BLOCK[12,BYTE], ! String descriptor for current lexeme
3109 4745 2 LEX_BUF : BLOCK[CHS_PER_LEXEME,BYTE], ! Buffer to hold current lexeme
3110 4746 2 OPR_FLAG; ! Operand flag
3111 4747 2
3112 4748 2 LITERAL
3113 4749 2 SPACE = %X'20'; ! ASCII value for space
3114 4750 2 COMMA = %X'2C'; ! ASCII value for comma
```

```
3115 4751 2
3116 4752 2
3117 4753 2
3118 4754 2
3119 4755 2
3120 4756 2
3121 4757 2
3122 4758 2
3123 4759 2
3124 4760 2
3125 4761 2
3126 4762 2
3127 4763 2
3128 4764 2
3129 4765 2
3130 4766 2
3131 4767 2
3132 4768 2
3133 4769 2
3134 4770 2
3135 4771 3
3136 4772 3
3137 4773 3
3138 4774 3
3139 4775 3
3140 4776 3
3141 4777 3
3142 4778 3
3143 4779 3
3144 4780 4
3145 4781 4
3146 4782 4
3147 4783 4
3148 4784 4
3149 4785 5
3150 4786 4
3151 4787 4
3152 4788 4
3153 4789 4
3154 4790 4
3155 4791 4
3156 4792 4
3157 4793 4
3158 4794 4
3159 4795 3
3160 4796 3
3161 4797 2
3162 4798 2
3163 4799 2
3164 4800 2
3165 4801 2
3166 4802 2
3167 4803 2
3168 4804 2
3169 4805 2
3170 4806 3
3171 4807 3

!++
! Define a string descriptor for the instruction to be reduced.
--
INS_STG_DESC[DSC$W_LENGTH] = .INS_PTR[0];
INS_STG_DESC[DSC$A_POINTER] = .INS_PTR[1];
INS_PTR[.INS_PTR[0]+1] = 0; ! Set an end-of-line indicator

!++
! Get the first lexeme which will be either a label or the opcode.
! Put it into the output buffer. Then search for the next significant
! character. If it is a colon, then move the colon into the output buffer
! and get the opcode and move it in also.
--
LABEL_FLAG = TRUE;
EDITED_INS_DESC[DSC$W_LENGTH] = 0;
LEXEME_DESC[DSC$A_POINTER] = LEX_BUF;
LEXEME_DESC[DSC$W_MAXLEN] = CHS_PER_LEXEME;
REPEAT
  BEGIN
    LEXEME_DESC[DSC$W_LENGTH] = 0;
    MAR_GET_LEX(INS_STG_DESC, LEXEME_DESC);
    CH$COPYT(LEXEME_DESC[DSC$W_LENGTH], .LEXEME_DESC[DSC$A_POINTER], SPACE,
      .LEXEME_DESC[DSC$W_LENGTH]+1,
      CH$PTR(.EDITED_INS_DESC[DSC$A_POINTER], .EDITED_INS_DESC[DSC$W_LENGTH]));
    EDITED_INS_DESC[DSC$W_LENGTH] = .EDITED_INS_DESC[DSC$W_LENGTH] +
      .LEXEME_DESC[DSC$W_LENGTH] + 1;
    IF .LABEL_FLAG
    THEN
      BEGIN
        INSTRUC_PTR = .INS_STG_DESC[DSC$A_POINTER];
        DO
          CHAR = CH$RCHAR_A(INSTRUC_PTR)
        UNTIL ((.CHAR NEQU ' ') AND (.CHAR NEQU ' '));
        IF (.CHAR NEQU ':')
        THEN
          EXITLOOP;
        CH$MOVE(1, CHAR, CH$PTR(.EDITED_INS_DESC[DSC$A_POINTER], .EDITED_INS_DESC[DSC$W_LENGTH]));
        EDITED_INS_DESC[DSC$W_LENGTH] = .EDITED_INS_DESC[DSC$W_LENGTH] + 1;
        INS_STG_DESC[DSC$W_LENGTH] = .INS_STG_DESC[DSC$W_LENGTH] +
          .INS_STG_DESC[DSC$A_POINTER] - .INSTRUC_PTR;
        INS_STG_DESC[DSC$A_POINTER] = .INSTRUC_PTR;
        LABEL_FLAG = FALSE;
      END
    ELSE
      EXITLOOP;
    END;
  END;
!++
! Now loop to get each operand. The call to GET_OPERAND reduces all
! the expressions and symbols in the operand to absolute numbers.
! The only exception is user defined symbols. These remain un-reduced
! to allow correct use of PATCH area addresses in the command file.
--
REPEAT
  BEGIN
    LEXEME_DESC[DSC$W_LENGTH] = 0;
```

3172 4808 3
3173 4809 4
3174 4810 3
3175 4811 4
3176 4812 4
3177 4813 4
3178 4814 4
3179 4815 4
3180 4816 4
3181 4817 3
3182 4818 3
3183 4819 3
3184 4820 3
3185 4821 3
3186 4822 3
3187 4823 4
3188 4824 3
3189 4825 3
3190 4826 4
3191 4827 3
3192 4828 3
3193 4829 4
3194 4830 4
3195 4831 3
3196 4832 3
3197 4833 3
3198 4834 3
3199 4835 3
3200 4836 3
3201 4837 2
3202 4838 1 END;

```
LEXEME_DESC[DSC$A_POINTER] = LEX_BUF;
IF NOT (OPR_FLAG = GET_OPERAND(INS_STG_DESC, LEXEME_DESC, FALSE, 0))
THEN
    BEGIN
        !++
        ! No more operands. Return successfully.
        !--
        EDITED_INS_DESC[DSC$W_LENGTH] = .EDITED_INS_DESC[DSC$W_LENGTH] - 1;
        RETURN;
    END;

!++
! If this is an operand containing an user-defined symbol, then
! the string ends with a comma or null. Ignore this character.
!--
IF (.OPR_FLAG EQLU OPR_FORW_REF)
THEN
    LEXEME_DESC[DSC$W_LENGTH] = .LEXEME_DESC[DSC$W_LENGTH] - 1;
    IF (.LEXEME_DESC[DSC$W_LENGTH] + .EDITED_INS_DESC[DSC$W_LENGTH] + 1)
        GTR .EDITED_INS_DESC[DSC$W_MAXLEN]
    THEN
        BEGIN
            SIGNAL(PAT$OUTCMDLNG+MSG$K_SEVERE);
        END;
    CH$COPY(.LEXEME_DESC[DSC$W_LENGTH], .LEXEME_DESC[DSC$A_POINTER], COMMA,
        .LEXEME_DESC[DSC$W_LENGTH]+1,
        CH$PTR(.EDITED_INS_DESC[DSC$A_POINTER], .EDITED_INS_DESC[DSC$W_LENGTH]));
    EDITED_INS_DESC[DSC$W_LENGTH] = .EDITED_INS_DESC[DSC$W_LENGTH] +
        .LEXEME_DESC[DSC$W_LENGTH] + 1;
END;
```

! End of PAT\$REDUCE_INS

				03FC 00000	.ENTRY	PAT\$REDUCE_INS, Save R2,R3,R4,R5,R6,R7,R8,-		
						R9	4694	
					SUBL2	#48, SP		
		28	AE	04	BC	9B 00005	MOVZBW @INS_PTR, INS_STG_DESC	4755
2C	AE	04	AC		01	C1 0000A	ADDL3 #1, INS_PTR, INS_STG_DESC+4	4756
			50	04	BC	9A 00010	MOVZBL @INS_PTR, R0	4757
			50	04	AC	C0 00014	ADDL2 INS_PTR, R0	
				01	A0	94 00018	CLRB 1(R0)	
			59		01	D0 0001B	MOVL #1, LABEL_FLAG	4765
			56	08	AC	D0 0001E	MOVL EDITED_INS_DESC, R6	4766
					66	B4 00022	CLRW (R6)	
		20	AE		6E	9E 00024	MOVAB LEX_BUF, LEXEME_DESC+4	4767
		24	AE		19	B0 00028	MOVW #25, LEXEME_DESC+8	4768
				1C	AE	B4 0002C	CLRW LEXEME_DESC	4771
				1C	AE	9F 0002F	PUSHAB LEXEME_DESC	4772
				2C	AE	9F 00032	PUSHAB INS_STG_DESC	
FF5C	CF			02	FB	00035	CALLS #2, MAR_GET_LEX	
	51		1C	AE	3C	0003A	MOVZWL LEXEME_DESC, R1	4774
				51	D6	0003E	INCL R1	
	50			66	3C	00040	MOVZWL (R6), R0	4775
	50		04	A6	C0	00043	ADDL2 4(R6), R0	

51	20	20	BE	1C	AE	2C	00C47	MOVCS	LEXEME_DESC, @LEXEME_DESC+4, #32, R1, (R0)	
					60		0004E			
			50		66	3C	0004F	MOVZWL	(R6), R0	4777
			51	1C	AE	3C	00052	MOVZWL	LEXEME_DESC, R1	
			50		51	C0	00056	ADDL2	R1, R0	
	66		50		01	A1	00059	ADDW3	#1, R0, (R6)	
			37		59	E9	0005D	BLBC	LABEL_FLAG, 3\$	4778
			57	2C	AE	D0	00060	MOVL	INS_STG_DESC+4, INSTRU_PTR	4781
			58		87	90	00064	MOVB	(INSTRU_PTR)+, CHAR	4783
			20		58	91	00067	CMPB	CHAR, #32	4784
					F8	13	0006A	BEQL	2\$	
			09		58	91	0006C	CMPB	CHAR, #9	
					F3	13	0006F	BEQL	2\$	
			3A		58	91	00071	CMPB	CHAR, #58	4785
					21	12	00074	BNEQ	3\$	
			50		66	3C	00076	MOVZWL	(R6), R0	4788
			50	04	A6	C0	00079	ADDL2	4(R6), R0	
			60		58	90	0007D	MOVB	CHAR, (R0)	
					66	B6	00080	INCW	(R6)	4789
			50	28	AE	3C	00082	MOVZWL	INS_STG_DESC, R0	4791
			50	2C	AE	C0	00086	ADDL2	INS_STG_DESC+4, R0	
28	AE		50		57	A3	0008A	SUBW3	INSTRU_PTR, R0, INS_STG_DESC	
		2C	AE		57	D0	0008F	MOVL	INSTRU_PTR, INS_STG_DESC+4	4792
					59	D4	00093	CLRL	LABEL_FLAG	4793
					95	11	00095	BRB	1\$	4778
				1C	AE	B4	00097	CLRW	LEXEME_DESC	4807
		20	AE		6E	9E	0009A	MOVAB	LEX_BUF, LEXEME_DESC+4	4808
					7E	7C	0009E	CLRQ	-(SP)	4809
				24	AE	9F	000A0	PUSHAB	LEXEME_DESC	
				34	AE	9F	000A3	PUSHAB	INS_STG_DESC	
		F330	CF		04	FB	000A6	CALLS	#4, GET_OPERAND	
			57		50	D0	000AB	MOVL	R0, OPR_FLAG	
			03		57	E8	000AE	BLBS	OPR_FLAG, 4\$	
					66	B7	000B1	DECW	(R6)	4815
						04	000B3	RET		4811
			03		57	D1	000B4	CMPL	OPR_FLAG, #3	4823
					03	12	000B7	BNEQ	5\$	
				1C	AE	B7	000B9	DECW	LEXEME_DESC	4825
			50	1C	AE	3C	000BC	MOVZWL	LEXEME_DESC, R0	4826
			51		66	3C	000C0	MOVZWL	(R6), R1	
			50		51	C0	000C3	ADDL2	R1, R0	
					50	D6	000C6	INCL	R0	
50	08	A6	10		00	ED	000C8	CMPZV	#0, #16, 8(R6), R0	4827
					0D	18	000CE	BGEQ	6\$	
					8F	DD	000D0	PUSHL	#7176842	4830
		00000000G	00		01	FB	000D6	CALLS	#1, LIB\$SIGNAL	
			51	1C	AE	3C	000DD	MOVZWL	LEXEME_DESC, R1	4833
					51	D6	000E1	INCL	R1	
			50		66	3C	000E3	MOVZWL	(R6), R0	4834
			50	04	A6	C0	000E6	ADDL2	4(R6), R0	
51	2C	20	BE	1C	AE	2C	000EA	MOVCS	LEXEME_DESC, @LEXEME_DESC+4, #44, R1, (R0)	
					60		000F1			
			50		66	3C	000F2	MOVZWL	(R6), R0	4836
			51	1C	AE	3C	000F5	MOVZWL	LEXEME_DESC, R1	
			50		51	C0	000F9	ADDL2	R1, R0	
		66	50		01	A1	000FC	ADDW3	#1, R0, (R6)	
					95	11	00100	BRB	3\$	4797

PATENC
V04-000

C 2
16-Sep-1984 00:40:15
14-Sep-1984 12:52:32

VAX-11 Bliss-32 V4.0-742
DISK\$VMMASTER:[PATCH.SRC]PATENC.B32;1 (14)

Page 93

PAT
V04

; Routine Size: 258 bytes, Routine Base: _PAT\$CODE + 1220

```
3204 4839 1 GLOBAL ROUTINE PAT$RESOLVE_INS (BUFFER_PTR) : NOVALUE =
3205 4840 1
3206 4841 1 !++
3207 4842 1 FUNCTIONAL DESCRIPTION:
3208 4843 1
3209 4844 1     This routine resolves all the forward references in the FWR table.
3210 4845 1     It encodes the operands and enters them into the appropriate bytes
3211 4846 1     in the buffers.
3212 4847 1
3213 4848 1 FORMAL PARAMETERS:
3214 4849 1
3215 4850 1     BUFFER_PTR - Address of the descriptor for the buffer holding the
3216 4851 1                 binary instruction stream to be resolved
3217 4852 1
3218 4853 1 IMPLICIT INPUTS:
3219 4854 1
3220 4855 1     PAT$GL_FWRLHD - Forward Reference table listhead
3221 4856 1
3222 4857 1 IMPLICIT OUTPUTS:
3223 4858 1
3224 4859 1     The resolved operands are witten into the instruction byte streams.
3225 4860 1
3226 4861 1 ROUTINE VALUE:
3227 4862 1
3228 4863 1     FALSE - if any resolution fails.
3229 4864 1     TRUE - if all resolutions were successful.
3230 4865 1
3231 4866 1 COMPLETION CODES:
3232 4867 1
3233 4868 1     NONE
3234 4869 1
3235 4870 1 SIDE EFFECTS:
3236 4871 1
3237 4872 1     none
3238 4873 1
3239 4874 1 --
3240 4875 1
3241 4876 2 BEGIN
3242 4877 2
3243 4878 2 MAP
3244 4879 2     BUFFER_PTR : REF BLOCK[,BYTE];
3245 4880 2
3246 4881 2 LOCAL
3247 4882 2     OUT_BYTE_PTR,
3248 4883 2     POINTER : REF BLOCK[,BYTE],
3249 4884 2     OPR_FLAG,
3250 4885 2     INST_STG_DESC : BLOCK [8, BYTE],
3251 4886 2     OPINFO_PTR : REF BLOCK[OPTSIZE, BYTE],
3252 4887 2     BRANCH_SIZE,
3253 4888 2     OPRND_STG_DESC : BLOCK[12,BYTE],
3254 4889 2     OPRND_BUF : VECTOR[MAX_BUF_SIZE,BYTE];
3255 4890 2
3256 4891 2 !++
3257 4892 2 ! Now loop, resolving each forward reference in the table.
3258 4893 2 !--
3259 4894 3 WHILE (.PAT$GL_FWRLHD NEQA 0)
3260 4895 2 DO
```

! String descriptor for binary instruction b

! Pointer into output buffer
! Pointer to forward reference table entry
! flag if operand found valid
! Instruction string descriptor
! Pointer into the OPINFO table

! String descriptor for next operand
! Buffer to hold next operand


```

3261 4896 3 BEGIN
3262 4897 3 POINTER = .PAT$GL_FWRLHD;
3263 4898 3 INST_STG_DESC[DSC$W_LENGTH] = .POINTER[FWR$W_OPRNDLNG];
3264 4899 3 INST_STG_DESC[DSC$a_pointer] = .POINTER[FWR$a_OPRNDADR];
3265 4900 3 OPRND_STG_DESC[DSC$W_LENGTH] = 0;
3266 4901 3 OPRND_STG_DESC[DSC$a_POINTER] = OPRND_BUF;
3267 4902 3 OPRND_STG_DESC[DSC$W_MAXLEN] = MAX_BUF_SIZE;
3268 4903 3 OPINFO_PTR = .POINTER[FWR$a_OPINFO];
3269 4904 3 OPR_FLAG = GET_OPERAND( INST_STG_DESC, OPRND_STG_DESC, TRUE, 1 ^ .OPINFO_PTR[OP_CONTEXT(.POINTER[FWR
3270 4905 4 IF (NOT .OPR_FLAG) OR (.OPR_FLAG EQLU OPR_FORW_REF)
3271 4906 3 THEN
3272 4907 3     SIGNAL(PAT$INVOPRND+MSG$K_WARN, 2, .POINTER[FWR$W_OPRNDLNG], .POINTER[FWR$a_OPRNDADR]);
3273 4908 3
3274 4909 3 ++
3275 4910 3 | Extract and encode one operand reference. Give up if this fails.
3276 4911 3 --
3277 4912 3 BRANCH_SIZE = NO_BR;
3278 4913 3 OUT_BYTE_PTR = CR$PTR(.BUFFER_PTR[DSC$a_POINTER], .POINTER[FWR$B_BUFOFF]);
3279 4914 3 IF (.OPINFO_PTR[OP_NUMOPS] EQL .POINTER[FWR$B_NTHOPRND]) OR
3280 4915 4 (.OPINFO_PTR[OP_NUMOPS] EQL ASM_DIR_OP)
3281 4916 3 THEN
3282 4917 3     IF (.POINTER[FWR$B_NTHOPRND] LSS MAXOPRND) OR
3283 4918 4 (.OPINFO_PTR[OP_NUMOPS] EQL ASM_DIR_OP)
3284 4919 3 THEN
3285 4920 4 BEGIN
3286 4921 4     BRANCH_SIZE = .OPINFO_PTR[OP_BR_TYPE];
3287 4922 4     IF (.BRANCH_SIZE EQLU BR_LG)
3288 4923 4 THEN
3289 4924 4         BRANCH_SIZE = A_LONGWORD;
3290 4925 3     END;
3291 4926 4 IF NOT (PAT$GL_ERRCODE = ENC_OPERAND( OPRND_STG_DESC, OUT_BYTE_PTR,
3292 4927 4     T ^ .OPINFO_PTR[OP_CONTEXT(.POINTER[FWR$B_NTHOPRND]) ],
3293 4928 4     .BRANCH_SIZE, .POINTER[FWR$L_PC], .OPINFO_PTR))
3294 4929 3 THEN
3295 4930 4 BEGIN
3296 4931 4 ++
3297 4932 4 | "Operand Syntax" error. Instruction substitution cannot be
3298 4933 4 | done as it would alter all the offsets in the forward Reference
3299 4934 4 | table (FWR$B_BUFOFF fields) and the remaining resolutions
3300 4935 4 | would be written into the wrong places in the buffer.
3301 4936 4 --
3302 4937 5 IF (.PAT$GL_ERRCODE EQL PAT$K_BR_RANGE)
3303 4938 4 THEN
3304 4939 4     SIGNAL(PAT$BRT00FAR+MSG$K_INFO);
3305 4940 4     SIGNAL(PAT$INVOPRND+MSG$K_WARN, 2, .POINTER[FWR$W_OPRNDLNG], .POINTER[FWR$a_OPRNDADR])
3306 4941 3 END;
3307 4942 4 IF (.OPRND_STG_DESC[DSC$W_LENGTH] NEQ 0)
3308 4943 3 THEN
3309 4944 3     SIGNAL(PAT$INVOPRND+MSG$K_WARN, 2, .POINTER[FWR$W_OPRNDLNG],
3310 4945 3     .POINTER[FWR$a_OPRNDADR]);
3311 4946 3 PAT$GL_FWRLHD = .POINTER[FWR$L_FLINK];
3312 4947 3 PAT$FREERELEASE(.POINTER, (FWR$C_SIZE + 3)/4);
3313 4948 2 END;
3314 4949 2 RETURN;
3315 4950 1 END;
```

! End of PAT\$RESOLVE_INS

				07FC 00000	.ENTRY	PAT\$RESOLVE_INS, Save R2,R3,R4,R5,R6,R7,R8,-;		
				5A 00000000G	EF 9E 00002	MOVAB	R9,R10	4839
				59 00000000G	EF 9E 00009	MOVAB	PAT\$GL_FWRLHD, R10	
				58 00000000G	00 9E 00010	MOVAB	PAT\$GL_ERRCODE, R9	
				5E FDEB	CE 9E 00017	MOVAB	LIB\$SIGNAL, R8	
				55 04	AC D0 0001C	MOVAB	-536(SP), SP	
				50	6A D0 00020	MOVL	BUFFER_PTR, R5	4913
					01 12 00023	MOVL	PAT\$GL_FWRLHD, R0	4894
					04 0C 025	BNEQ	2\$	
				52	50 D0 0C 026	RET		
				F8 AD 08	A2 B0 00029	MOVL	R0, POINTER	4897
				FC AD 0C	A2 D0 0002E	MOVW	8(POINTER), INST_STG_DESC	4898
					EC AD B4 00033	MOVL	12(POINTER), INST_STG_DESC+4	4899
				F0 AD 04	AE 9E 00036	CLRW	OPRND_STG_DESC	4900
				F4 AD 0200	8F B0 0003B	MOVAB	OPRND_BUF, OPRND_STG_DESC+4	4901
				53 10	A2 D0 00041	MOVW	#512, OPRND_STG_DESC+8	4902
				51 08	A2 9A 00045	MOVL	16(POINTER), OPINFO_PTR	4903
				51	04 C4 00049	MOVZBL	11(POINTER), R1	4904
				04	51 EF 0004C	MULL2	#4, R1	
50	04	A3		01	50 78 00052	EXTZV	R1, #4, 4(OPINFO_PTR), R0	
					54 DD 00056	ASHL	R0, #1, R4	
					01 DD 00058	PUSHL	R4	
					EC AD 9F 0005A	PUSHL	#1	
				F8 AD 9F 0005D	PUSHAB	OPRND_STG_DESC		
				F274 CF 04 FB 00060	PUSHAB	INST_STG_DESC		
				57 50 D0 00065	CALLS	#4, GET_OPERAND		
				05 57 E9 00068	MOVL	R0, OPR_FLAG		
				03 57 D1 0006B	BLBC	OPR_FLAG, 3\$	4905	
					12 12 0006E	CMPL	OPR_FLAG, #3	
					OC A2 DD 00070	BNEQ	4\$	
				7E 08 A2 3C 00073	PUSHL	12(POINTER)	4907	
					02 DD 00077	MOVZWL	8(POINTER), -(SP)	
				006D8278 8F DD 00079	PUSHL	#2		
				68 04 FB 0007F	PUSHL	#7176824		
					56 D4 00082	CALLS	#4, LIB\$SIGNAL	
					A2 C1 00084	CLRL	BRANCH_SIZE	4912
		6E	04	50 08 A2 9A 0008A	ADDL3	20(POINTER), 4(R5), OUT_BYTE_PTR	4913	
				04 00 EC 0008E	MOVZBL	11(POINTER), R0	4914	
					0C 13 00094	CMPL	#0, #4, 4(OPINFO_PTR), R0	
					00 EC 00096	BEQL	5\$	
FFFFFFFE	8F	04	A3	04 20 12 000A0	CMPL	#0, #4, 4(OPINFO_PTR), #2	4915	
				06 08 A2 91 000A2	BNEQ	7\$		
					0C 1F 000A6	CMPL	11(POINTER), #6	4917
					00 EC 000A8	BLSSU	6\$	
FFFFFFFE	8F	04	A3	04 0E 12 000B2	CMPL	#0, #4, 4(OPINFO_PTR), #2	4918	
					04 EF 000B4	BNEQ	7\$	
				03 56 D1 000BA	EXTZV	#4, #4, 7(OPINFO_PTR), BRANCH_SIZE	4921	
					03 12 000BD	CMPL	BRANCH_SIZE, #3	4922
				56 04 D0 000BF	BNEQ	7\$		
					53 DD 000C2	MOVL	#4, BRANCH_SIZE	4924
					04 A2 9A 000C4	PUSHL	OPINFO_PTR	4928
				04 8F BB 000C7	PUSHAB	4(POINTER)		
				0050 AE 9F 000CB	PUSHR	#*M<R4,R6>		
				10		PJSHAB	OUT_BYTE_PTR	4926

F881	CF	EC	AD	9F	000C	PUSHAB	OPRND STG_DESC	:	
	69		G6	FB	000D	CALLS	#6, ENC_OPERAND	:	4928
	20		50	D0	000D	MOVL	R0, PAT\$GL_ERRCODE	:	
	02		50	E8	000D	BLBS	R0, 9\$	:	
			69	D1	000D	CMPL	PAT\$GL_ERRCODE, #2	:	4937
			09	12	000D	BNEQ	8\$	:	
		006D8223	8F	DD	000E	PUSHL	#7176739	:	4939
	68		01	FB	000E	CALLS	#1, LIB\$SIGNAL	:	
		0C	A2	DD	000E	PUSHL	12(POINTER)	:	4940
	7E	08	A2	3C	000E	MOVZWL	8(POINTER), -(SP)	:	
			02	DD	000F	PUSHL	#2	:	
		006D8278	8F	DD	000F	PUSHL	#7176824	:	
	68		04	FB	000F	CALLS	#4, LIB\$SIGNAL	:	
		EC	AD	B5	000F	TSTW	OPRND_STG_DESC	:	4942
			12	13	000F	BEQL	10\$	:	
		0C	A2	DD	0010	PUSHL	12(POINTER)	:	4945
	7E	08	A2	3C	0010	MOVZWL	8(POINTER), -(SP)	:	4944
			02	DD	0010	PUSHL	#2	:	
		006D8278	8F	DD	0010	PUSHL	#7176824	:	
	68		04	FB	0011	CALLS	#4, LIB\$SIGNAL	:	
	6A		62	D0	0011	MOVL	(POINTER), PAT\$GL_FWRLHD	:	4946
			06	DD	0011	PUSHL	#6	:	4947
			52	DD	0011	PUSHL	POINTLR	:	
00000000G	EF		02	FB	0011	CALLS	#2, PAT\$FREERELEASE	:	
			FEFC	31	0012	BRW	1\$	:	4894
				04	0012	RET		:	4950

; Routine Size: 293 bytes, Routine Base: _PAT\$CODE + 132E

: 3317 4951 1 END
: 3318 4952 0 ELUDOM

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes					
PAT\$OWN	12	NOVEC,	WRT,	RD	NOEXE,NOSHR,	LCL,	REL, CON,NOPIC,ALIGN(2)
PAT\$CODE	5203	NOVEC,NOWRT,	RD		EXE,NOSHR,	LCL,	REL, CON,NOPIC,ALIGN(2)
ABS	0	NOVEC,NOWRT,NORD			NOEXE,NOSHR,	LCL,	ABS, CON,NOPIC,ALIGN(0)
PAT\$PLIT	6	NOVEC,NOWRT,	RD		NOEXE,NOSHR,	LCL,	REL, CON,NOPIC,ALIGN(0)

Library Statistics

File	----- Symbols -----		Pages Mapped	Processing Time
	Total	Loaded Percent		
_S255\$DUA28:[SYSLIB]LIB.L32;1	18619	7 0	1000	00:01.9

: Information: 1
: Warnings: 0
: Errors: 0

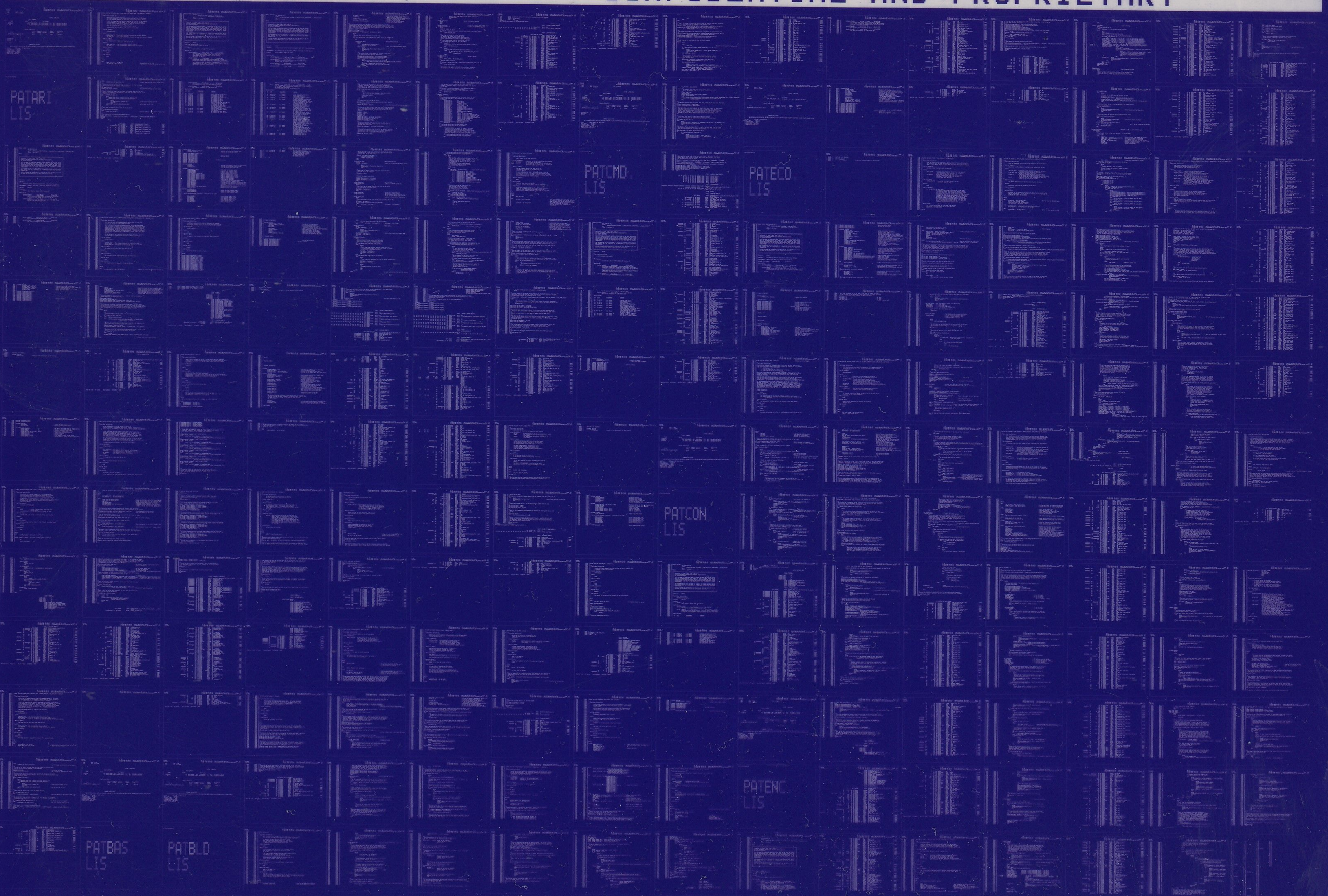
COMMAND QUALIFIERS

: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/VARIANT:1/LIS=LISS:PATENC/OBJ=OBJ\$:PATENC MSRCS:PATENC/UPDATE=(ENHS:PATENC)

: Size: 5203 code + 18 data bytes
: Run Time: 01:44.7
: Elapsed Time: 05:52.8
: Lines/CPU Min: 2837
: Lexemes/CPU-Min: 22673
: Memory Used: 506 pages
: Compilation Complete

0300 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



0301 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY